GUIDE DU BASIC

ZX 81

Douglas Hergert





GUIDE DU BASIC

ZX 81

A Robert Weinberg

Traduction de **Nellie Saumont**
Illustration de **Jean-François Pénichoux**

ISBN 2-902414-85-4

Photocompsition RCV - Paris
(Édition originale :
ISBN 0-89588-113-6 Sybex Inc. Berkeley)

GUIDE DU BASIC ZX 81

Douglas Hergert



Paris – Berkeley – Düsseldorf

Table des matières

M

N

O

P

R

S

INTRODUCTION

Alors que dans le monde de la micro-informatique, on attendait avec impatience la sortie du ZX 81, on n'a guère pris au sérieux l'impact qu'il pouvait avoir sur les nouveaux utilisateurs d'ordinateur. Certains l'ont relégué, d'un point de vue technique, au rang d'un jouet, d'un ordinateur n'ayant pour intérêt que l'utilisation de jeux électroniques. (D'autres, plus cyniques, ont même nié sa valeur ludique.) Mais en général, on a pris conscience de l'importance potentielle du ZX 81 en tant qu'outil éducatif.

Ceux qui critiquent ce petit ordinateur ont toutefois raison sur un point : il serait décevant d'attendre trop des utilisations courantes du ZX 81. Si l'on veut pouvoir utiliser le traitement de texte, des programmes de gestion comptable ou le stockage et la manipulation de données en mémoire de masse, il faut investir des sommes plus importantes dans l'achat d'un ordinateur. D'autre part, bien des gens achètent leur premier ordinateur pour satisfaire leur désir d'élargir leurs connaissances : pour apprendre à écrire des programmes en BASIC et plus généralement pour maîtriser quelques-uns des principes essentiels de la programmation. Dans ce cas, si l'on veut limiter l'investissement, le ZX 81 est sans nul doute le meilleur choix. Sa version du BASIC, bien que particulière dans certains détails, n'est toutefois qu'une variante des BASIC que l'on trouve dans la plupart des ordinateurs individuels, y compris dans ceux qui coûtent beaucoup plus cher. Son "hardware" est facile à manipuler et extrêmement fiable. Il coûte moins cher qu'un cours de programmation.

Ce livre est donc destiné à aider les utilisateurs du ZX 81 à maîtriser les éléments du BASIC. On y trouve l'ensemble du vocabulaire de cet ordinateur. Les rubriques ont une présentation pratique et un style clair qui les rendent faciles à comprendre et à utiliser. Chaque rubrique comporte :

- La description d'un mot clé (ou d'une fonction) BASIC — ce qu'il fait et comment l'utiliser correctement dans un programme

- Un exemple de programme et une explication de son fonctionnement en illustration du mot ou de la fonction BASIC

- Un affichage de résultats à partir de l'exemple de programme

- Des "Notes et commentaires". Un ensemble de remarques et d'informations pratiques — par exemple : propositions d'utilisations supplémentaires, mise en garde contre des erreurs possibles, renvois à d'autres mots BASIC en rapport avec le mot étudié.

Cette présentation varie occasionnellement pour répondre aux impératifs propres à chaque rubrique. Dans tous les cas, le but est d'intégrer les ordres BASIC au vocabulaire *appliqué* à la programmation afin qu'ils puissent être immédiatement employés par l'utilisateur pour la création de ses propres programmes.

On portera une attention particulière aux exemples de programmes du livre. Les éléments d'un langage de programmation sont plus faciles à retenir à l'aide d'exemples concrets qu'à partir d'une description abstraite. Pour tirer un profit maximum de ces exemples, il est nécessaire d'entrer ces programmes dans l'ordinateur et de les faire tourner. Chaque programme est étudié pour illustrer les caractéristiques, les possibilités et parfois les subtilités d'une commande BASIC du ZX 81. Ces programmes sont des exercices et doivent être utilisés comme tels. Certains d'entre eux peuvent d'ailleurs être utiles et même amusants. On trouve par exemple parmi les programmes de ce livre :

- Un programme qui crée des histogrammes à partir de données numériques que l'on entre par le clavier (voir la rubrique DIM).

- Un programme qui aide à gérer son compte en banque (voir la rubrique GOTO).

- Quelques programmes d'entrée de données facilement utilisables (voir les rubriques INKEY $, VAL).

- Des programmes qui permettent de découvrir l'organisation de la mémoire de l'ordinateur (voir les rubriques PEEK, POKE).

- Des programmes qui affichent différentes sortes de graphiques à l'écran (voir les rubriques PLOT, STEP).

- Deux sous-programmes qui sont la base de tout jeu de cartes informatisé (voir les rubriques RND, TO).

- Un programme qui convertit des valeurs numériques en francs et en centimes et les affiche sous forme de chaînes de caractères avec le mot "FRANCS", des virgules et des chiffres décimaux (voir la rubrique STR$).

Les rubriques de ce livre comportent, en plus des commandes BASIC, un certain nombre de termes informatiques généraux qu'il est nécessaire d'acquérir au cours de l'apprentissage de la programmation sur le ZX 81. La Figure 1 donne une liste de ces termes. Les définitions de ce livre évitent en général le jargon informatique inutile ; mais certains termes, comme ceux de cette liste, sont assez communément utilisés pour qu'il soit profitable de les apprendre. On trouvera, à la fin du livre, un index des commandes BASIC qui apparaissent dans les exemples de programmes.

Note :

Le titre de chaque rubrique contient des informations aidant à la localisation du mot sur le clavier du ZX 81 et à l'entrée de l'instruction dans l'ordinateur. Ces titres se présentent de la façon suivante :

NOM (mode d'introduction ; [touche])

Il y a trois *modes d'introduction* (c'est-à-dire trois façons d'entrer les instructions dans l'ordinateur) pour le vocabulaire BASIC du ZX 81.

1. *Mot clé.* On peut entrer un mot clé dès l'apparition, en vidéo inverse sur l'écran, du curseur K. Chaque ligne d'un programme BASIC écrit pour le ZX 81 doit commencer par un mot clé. Les mots clés se trouvent sur le clavier au-dessus des touches lettres.

2. *Fonction.* Il faut, pour entrer une fonction, commencer par mettre l'ordinateur en mode fonction. On doit appuyer sur deux touches en même temps : d'abord sur la touche [SHIFT] (située dans l'angle inférieur gauche du clavier), puis sur la touche [ENTER] (tout à fait à droite du clavier). Le curseur F apparaît alors en vidéo inverse sur l'écran. Cela signifie que l'ordinateur est prêt à accepter l'entrée de fonctions. Celles-ci se trouvent sur le clavier au-dessous des touches lettres.

3. *Shift.* Les caractères obtenus grâce à la touche [SHIFT] sont représentés en rouge sur le clavier. Certains d'entre eux sont des signes de ponctuation ou des noms de fonctions. Il faut, pour entrer l'un de ces mots, appuyer simultanément sur la touche [SHIFT] et sur la touche portant le mot que l'on veut entrer. Cinq de ces mots sont de véritables mots clés que l'on ne doit entrer qu'après l'affichage du curseur K à l'écran. Ce sont les mots STOP, LPRINT, SLOW, FAST, et LLIST (situés respectivement sur les touches [A], [S], [D], [F], [G]). Les cinq autres ne sont pas des mots clés et ne doivent jamais apparaître au début d'une instruction BASIC. On peut entrer ces mots après l'affichage, en vidéo inverse, du curseur L. Ce sont les mots AND, THEN, et OR (situés respectivement sur les touches [2], [3] et [W]) qui sont associés à la syntaxe de l'instruction IF et les mots TO et STEP (touches [4] et [E]) qui sont associés à la syntaxe de l'instruction FOR.

La présentation des termes de vocabulaire général varie légèrement de celle des commandes BASIC :

Mot ou expression (vocabulaire informatique)

Le terme est en minuscules gras, seules les initiales sont en majuscules et la description entre parenthèses est seulement : "vocabulaire informatique" ou "vocabulaire informatique — ZX 81".

Algorithme	Ligne en cours
Argument	Menu
BASIC	Messages d'erreur
Chaîne	Notation scientifique
Code machine	Octet
Commande immédiate	Programme
Concaténation	Programmeur
Conversationnel	Sortie sur imprimante
Curseur	Sous-programme
Espace de travail	Tableau
Expression arithmétique	Utilitaires
Expression logique	Valeur littérale
Fonction	Variable

Figure 1 : Vocabulaire général de programmation décrit dans ce livre

ABS (fonction ; touche [G]

Cette fonction donne la valeur absolue d'un nombre. L'*argument* d'ABS peut être une valeur numérique littérale, une variable ou une expression arithmétique. ABS donne la valeur non signée du résultat.

Exemple de programme

ABS est utilisée lorsque l'on n'a pas besoin du signe (positif ou négatif) d'un nombre, comme c'est le cas dans le programme de la Figure A.1. Ce programme est un exercice simple dans lequel deux nombres sont choisis aléatoirement et comparés. Les deux nombres sont stockés dans les variables R1 et R2. La ligne 60 calcule et affiche leur différence :

```
60 PRINT " "; ABS (R1-R2);
```

```
   5 PRINT
  10 LET R1=RND
  20 LET R2=RND
  30 PRINT "PREMIER NOMBRE = ";R
1
  40 PRINT "SECOND NOMBRE  = ";R
2
  50 PRINT "==> LE PREMIER NOMBR
E EST, DE"
  60 PRINT "  ";ABS (R1-R2);" ,";
  70 IF R1<=R2 THEN PRINT " INFE
RIEUR "
  80 IF R1>R2 THEN PRINT " SUPER
IEUR "
  90 PRINT "  AU SECOND NOMBRE."
 100 PRINT
 105 PRINT
 110 GOTO 10
```

Figure A.1 : ABS-Exemple de programme

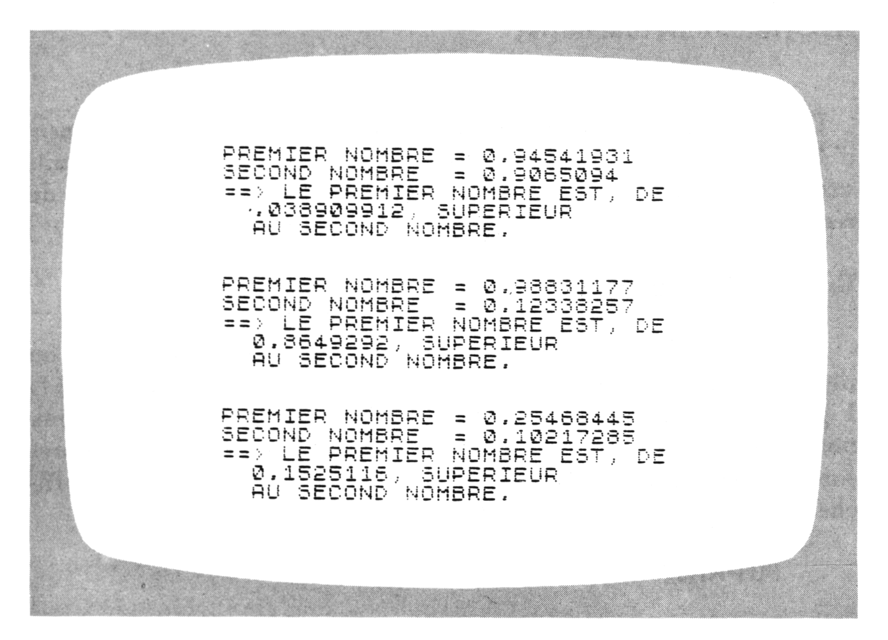


Figure A.2 : ABS-Exemple de résultat

Rien ne permet de savoir lequel des deux nombres sera le plus grand puisqu'ils sont pris au hasard. Par conséquent, l'expression :

```
R1-R2
```

peut donner un résultat positif ou négatif. Mais pour exprimer la différence entre R1 et R2, le signe est sans importance. On utilise donc la fonction ABS pour éliminer celui-ci. La Figure A.2. montre des résultats de ce programme qui donne, pour chaque couple de nombres pris au hasard, la valeur absolue de leur différence.

Algorithme (vocabulaire informatique)

Un algorithme est une série d'étapes à accomplir une tâche définie.

On peut commencer l'écriture d'un programme BASIC par l'élaboration d'un algorithme en langage naturel, avant d'essayer de la transcrire en une suite d'instructions BASIC. Considérons par exemple les étapes suivantes :

1. Augmenter la valeur de la variable V de 5.
2. Afficher la nouvelle valeur de V à l'écran.

Ces deux étapes peuvent être transcrites comme suit :

```
10 LET V=V+5
20 PRINT V
```

D'autre part, ce qui, exprimé en langage naturel, semble être un algorithme simple peut s'avérer beaucoup plus complexe une fois transcrit en BASIC. En voici un exemple :

1. Lire dix mots de 5 lettres à partir du clavier.
2. Ranger les mots par ordre alphabétique.
3. Afficher les mots à l'écran par ordre alphabétique.

Bien que ces étapes semblent simples et sans difficultés, on voit qu'elles nécessitent quinze instructions pour être exécutées avec succès sur un ordinateur :

```
 10 DIM W$(10,5)
 20 FOR I=1 TO 10
 30 INPUT W$ (I)
 40 NEXT I
 50 FOR I=1 TO 9
 60 FOR J=I+1 TO 10
 70 IF W$(I)<W$(J) THEN GOTO 110
 80 LET H$=W$(I)
 90 LET W$(I)=W$(J)
100 LET W$(J)=H$
110 NEXT J
120 NEXT I
130 FOR I=1  TO 10
140 PRINT W$(I)
150 NEXT I
```

Dans ce cas on peut trouver nécessaire de reprendre l'expression originale de l'algorithme et de réfléchir de manière plus approfondie aux différentes étapes avant d'entreprendre l'écriture du programme. Mais les trois étapes originales comprennent des tâches trop importantes pour que le BASIC puisse les réaliser à l'aide d'instructions simples. Prenons par exemple l'étape "Classer les mots par ordre alphabétique". Si le BASIC possédait l'instruction ALPHABETIZE dans son vocabulaire, il suffirait d'écrire l'instruction suivante :

```
50 ALPHABETIZE (W$)
```

Malheureusement, une telle instruction n'existe pas. Il faut donc trouver des instructions détaillées qui permettent à l'ordinateur de réaliser le classement alphabétique. On peut remplacer l'étape 2 par les étapes suivantes plus détaillées :

2a. Comparer chaque mot de la liste, un par un, avec chaque mot suivant.

2b. Si les deux mots ne sont pas classés alphabétiquement, alors les intervertir.

Ces étapes se rapprochent de la séquence d'instructions BASIC qu'il faut écrire ; cependant, les instructions ne sont pas encore suffisamment détaillées. Huit lignes de programme (50 à 120) sont nécessaires pour réaliser ces deux étapes. (Il faut remarquer que ces lignes caractérisent ce que l'on appelle un algorithme de *tri*.) Il faudra réexaminer l'algorithme original et trouver à quel niveau les étapes du processus ont été trop simplifiées. Par exemple, dans l'étape 2b, si l'on considère la phrase "intervertir leurs places dans la liste", le BASIC du ZX 81 ne possède pas l'instruction qui permette d'écrire :

```
80 SWAP (W$(I),W$(J))
```

On établira à la place les trois étapes nécessaires à l'inversion de l'ordre des mots :

2b(1). Stocker le premier mot dans une variable "de transition", H $.

2b(2). Stocker le second mot à la place initiale du premier.

2b(3). Stocker la valeur de H $ à la place initiale du second mot.

Ces trois étapes sont réalisées dans les trois lignes du programme numérotées de 80 à 100.

En résumé, il faut, par des "amplifications"successives, multiplier les étapes de l'algorithme afin qu'il soit assez détaillé pour permettre une transcription aisée en commandes BASIC.

AND (shift ; touche [2])

L'opérateur logique AND (ET) peut être utilisé pour créer une expression logique composée relative à une décision IF (SI).

La valeur ("vraie"ou "fausse") d'une telle expression composée dépend des valeurs des éléments combinés à AND. Une expression de la forme suivante :

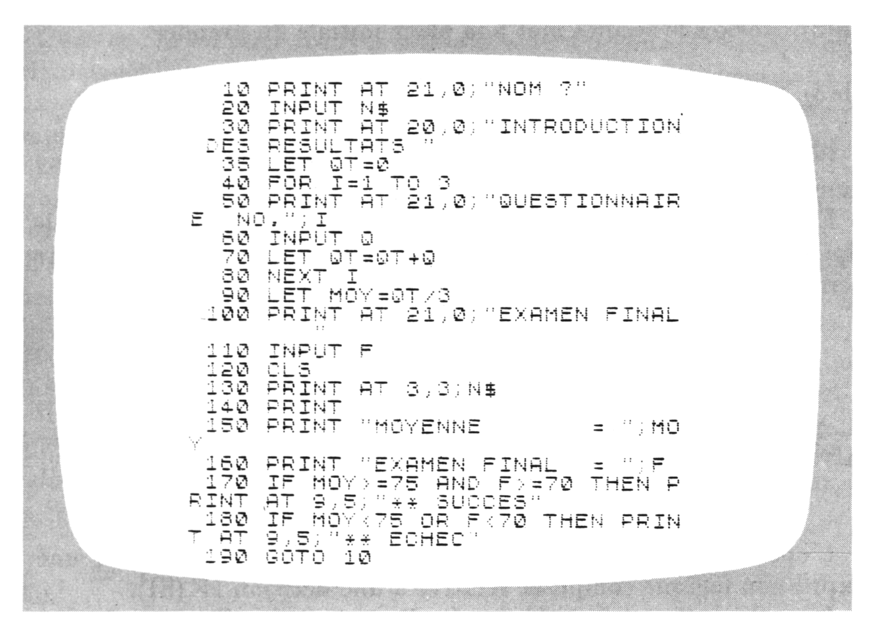
relation-1 **AND** relation-2

est vraie *seulement* si la relation-1 et la relation-2 sont vraies *toutes les deux*. Si l'une d'entre elles est fausse ou si les deux sont fausses, l'expression composée est fausse elle aussi.

Exemple de programme

Le programme de la Figure A.3 illustre l'utilisation de la fonction AND. Ce programme est conçu pour la situation hypothétique suivante : un professeur examine les résultats des épreuves semestrielles afin de voir quels sont les élèves qui ont réussi et quels sont ceux qui ont échoué.

Le professeur a donné trois questionnaires au cours du semestre et un examen final ; il a décidé que, pour réussir, un élève devait obtenir une moyenne au moins égale à 75 dans les questionnaires et un résultat d'au moins 70 à l'examen final. Pour utiliser ce programme, le professeur entre au clavier le nom de chaque élève et les résultats



```
  10 PRINT AT 21,0;"NOM ?"
  20 INPUT N$
  30 PRINT AT 20,0;"INTRODUCTION
 DES RESULTATS "
  35 LET QT=0
  40 FOR I=1 TO 3
  50 PRINT AT 21,0;"QUESTIONNAIR
E NO.";I
  60 INPUT Q
  70 LET QT=QT+Q
  80 NEXT I
  90 LET MOY=QT/3
 100 PRINT AT 21,0;"EXAMEN FINAL
"
 110 INPUT F
 120 CLS
 130 PRINT AT 3,3;N$
 140 PRINT
 150 PRINT "MOYENNE         = ";MO
Y
 160 PRINT "EXAMEN FINAL   = ";F
 170 IF MOY>=75 AND F>=70 THEN P
RINT AT 9,5;"** SUCCES"
 180 IF MOY<75 OR F<70 THEN PRIN
T AT 9,5;"** ECHEC"
 190 GOTO 10
```

Figure A.3 : AND-Exemple de programme

de ses épreuves ; l'ordinateur fera les calculs appropriés pour déterminer si l'élève a réussi ou non.

Les lignes 10 à 110 lisent les données relatives à un élève. Il faut remarquer en particulier la boucle FOR (lignes 40 à 80) qui lit le résultat de chaque questionnaire et l'additionne dans la variable QT. La ligne 90 attribue ensuite la moyenne des résultats des questionnaires à la variable MOY. La ligne 110 lit le résultat de l'examen final et l'attribue à la variable F.

Les lignes 130 à 180 affichent sur l'écran les renseignements relatifs à un élève. La ligne 170 illustre l'utilisation de la fonction AND :

> 170 **IF** MOY> =75 **AND** F> =70 **THEN**
> **PRINT AT** 9,5 ; "**SUCCES"

La fonction de l'instruction IF est d'afficher à l'écran le mot SUCCES si et seulement si *les deux* propositions suivantes sont vraies :

> MOY> =75
> F> =70

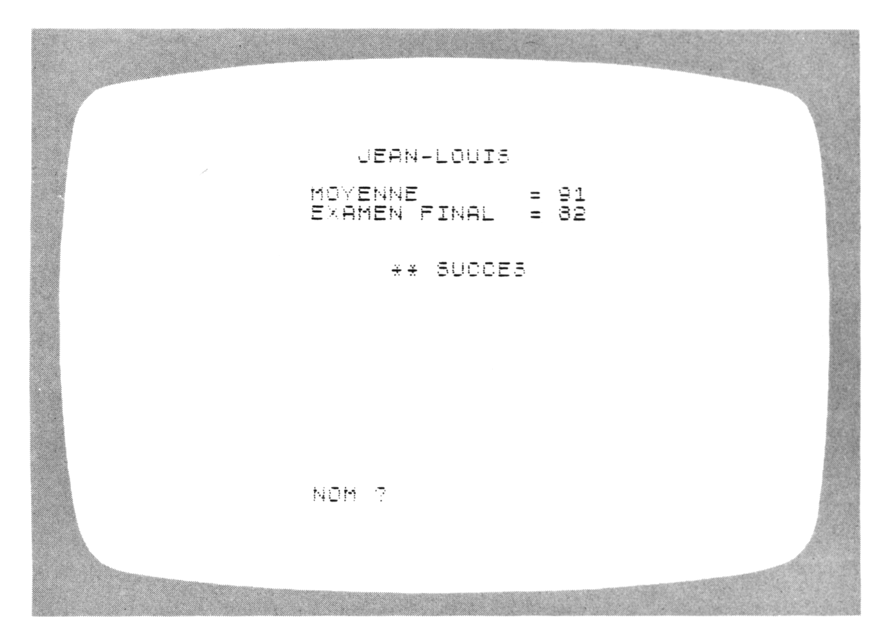


Figure A.4 : AND-Exemple de résultat (L'instruction composée est vraie.)

En d'autres termes, l'élève réussit seulement si la moyenne (MOY) de ses questionnaires est supérieure ou égale à 75 *et* (AND) si le résultat de son examen final (F) est supérieur ou égal à 70. Si l'une des conditions — ou les deux — ne sont pas remplies, la ligne 170 n'est pas exécutée et le programme continue à la ligne 180.

Les Figures A.4 et A.5 montrent les résultats de deux élèves. Le premier élève (Figure A.4) a réussi le semestre en satisfaisant aux deux conditions. Le second (Figure A.5) avait un résultat correct à l'examen final mais un résultat inférieur à 70 aux questionnaires, l'expression composée de la ligne 170 n'est donc pas valide et l'élève échoue.

Notes et commentaires

- La Figure A.6 est une "table de vérité"pour la condition ET (AND). Elle montre le résultat d'une relation composée, c'est-à-dire différentes combinaisons de valeurs pour les relations 1 et 2. On remarque que la relation composée est vraie seulement dans le cas où les relations internes sont *toutes les deux* vraies.

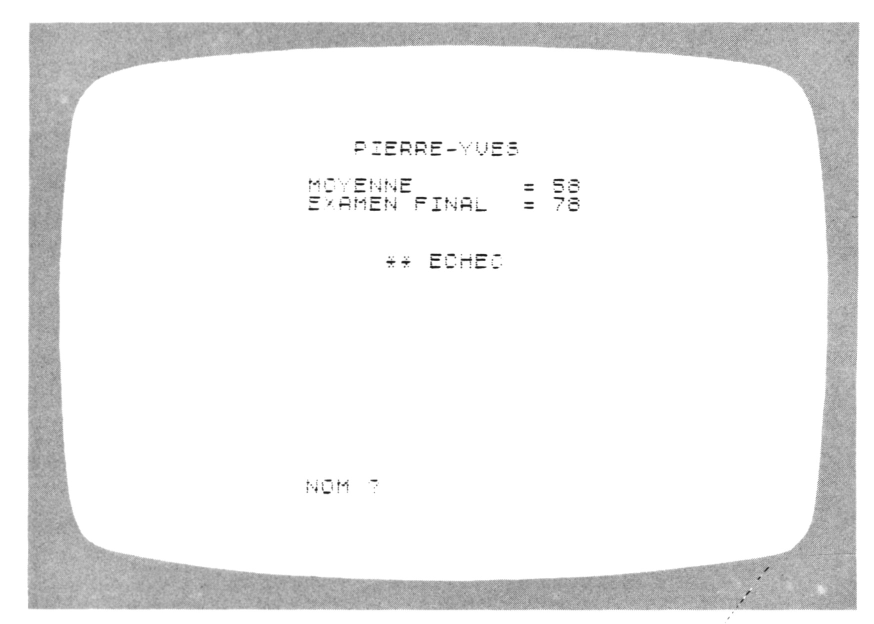


Figure A.5 : AND-Exemple de résultat (La relation composée est fausse.)

AND
===

TABLE DE VERITE

PROPOS. 1	PROPOS. 2	PROPOS. COMPOSEE
VRAIE	VRAIE	VRAIE
VRAIE	FAUSSE	FAUSSE
FAUSSE	VRAIE	FAUSSE
FAUSSE	FAUSSE	FAUSSE

Figure A.6 : AND-Table de vérité

- Les relations composées peuvent être constituées de plus de deux relations logiques. On peut utiliser des parenthèses pour spécifier l'ordre dans lequel les opérations doivent être effectuées. Par exemple, on suppose que notre professeur ait voulu prendre en compte un exercice scolaire d'un certain type :

  ```
  IF F> =70 AND (MOY> =75 OR EXERCICE> =80)
  THEN PRINT "SUCCES"
  ```

  La relation logique composée pour cette décision IF sera estimée vraie si et seulement si les deux conditions suivantes sont vérifiées :

  1. La variable F contient une valeur supérieure ou égale à 70, *et*

  2. Une ou les deux propositions suivantes sont vraies : MOY est supérieur ou égal à 75, et/ou EXERCICE est supérieur ou égal à 80.

- Pour plus d'information sur les relations logiques composées et sur les décisions IF, voir les rubriques IF, NOT, OR et "Expression logique."

ARCCOS (fonction ; touche [S])

La fonction ARCCOS donne l'arccosinus de n'importe quel nombre compris entre -1 et + 1. (L'arccosinus d'un nombre x est par définition l'angle dont le cosinus est x.) Le résultat de la fonction ARCCOS s'exprime en radians (et non en degrés).

Exemple de programme

La Figure A.7 montre un petit programme qui aide à étudier la fonction ARCCOS. On peut voir à la ligne 40 que la fonction ARCCOS est représentée à l'écran par ACS. La Figure A.8 montre le résultat de ce programme. On remarque que le résultat de la fonction

```
   5 PRINT
  10 PRINT "LA FONCTION ARCCOSIN
US"
  20 PRINT
  30 FOR I=-1 TO 1 STEP .25
  40 PRINT "ARCCOS(";I;")";TAB 1
5; " = ";ACS I
  50 NEXT I
```

Figure A.7 : ARCCOS-Exemple de programme

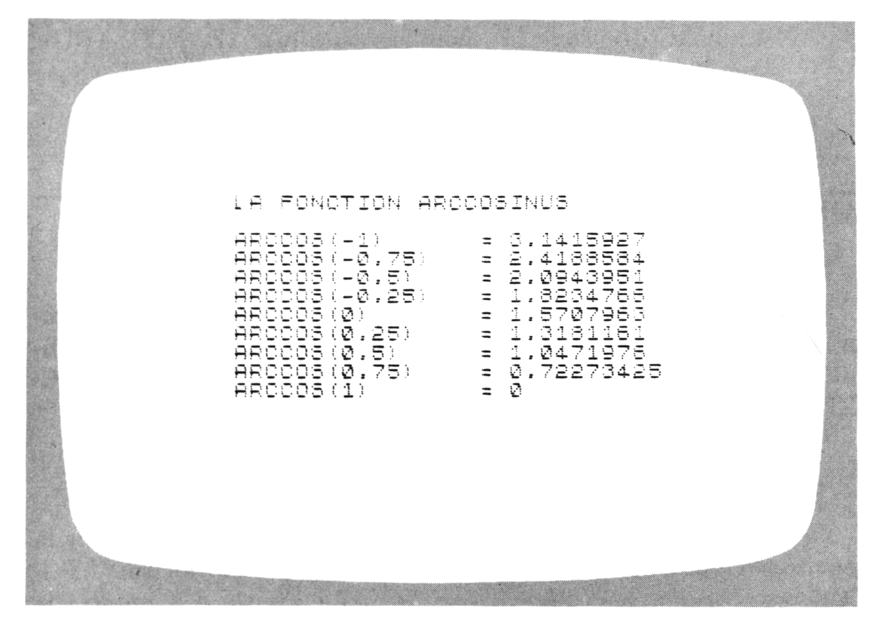


Figure A.8 : ARCCOS-Exemple de résultat

ARCCOS décroît de π à 0 quand l'argument augmente de -1 à +1. L'arccosinus de 0 est $\pi/2$.

Notes et commentaires

- La Figure A.9 montre une courbe de la fonction ARCCOS. (La courbe a été réalisée en utilisant l'instruction PLOT du ZX 81.) La fonction arccosinus est appelée en mathématiques *fonction multivaleurs*, car pour n'importe quelle valeur de x (argument de la fonction) on peut trouver de multiples valeurs de y (résultat de la fonction). La partie de la courbe comprise entre $y = 0$ et $y = \pi$ (comme dans la Figure A.9) est appelée *branche principale* de la fonction. La fonction ARCCOS en BASIC ZX 81 donne les valeurs de cette branche principale.

- Pour la conversion de radians en degrés, rappelons que 180 degrés équivalent à π radians. Un radian est donc égal à $180/\pi$, soit approximativement 57,3 degrés.

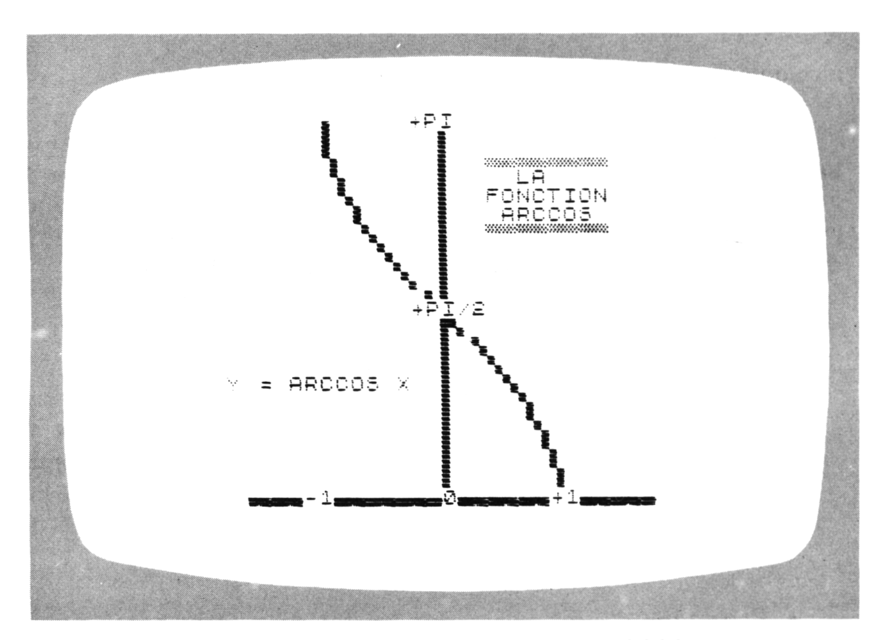


Figure A.9 : ARCCOS-Tracé de courbe

- L'attribution à la fonction ARCCOS d'un argument en dehors des limites -1 et +1 entraîne un message d'erreur signifiant que la fonction a été mal utilisée.

- Les autres fonctions trigonométriques inverses disponibles en BASIC ZX 81 sont ARCSIN et ARCTAN. Les fonctions trigonométriques sont SIN, COS et TAN.

ARCSIN (fonction ; touche [A])

La fonction ARCSIN donne l'arcsinus d'un nombre compris entre -1 et +1. (L'arcsinus d'un nombre x est par définition l'angle dont le sinus est x.) Le résultat de la fonction ARCSIN s'exprime en radians.

Exemple de programme

Le programme de la Figure A.10 illustre la fonction ARCSIN. La fonction ARCSIN est représentée sur l'écran par ASN comme le montre la ligne 40. La Figure A.11 montre le résultat de ce programme. Le résultat de la fonction ARCSIN croît de $-\pi/2$ à $+\pi/2$ quant l'argument varie de -1 à +1. L'arcsinus de 0 est 0.

Notes et commentaires

- La Figure A.12 montre la courbe de la fonction ARCSIN réalisée à l'aide de l'instruction PLOT du ZX 81. La fonction ARCSIN est, comme la fonction ARCCOS, une fonction *multivaleurs* ; la branche principale de la fonction est comprise entre $y = -\pi/2$ et $y = +\pi/2$. La fonction ARCSIN du BASIC du ZX 81 donne les valeurs de cette *branche principale.*

- Pour plus d'information sur les fonctions trigonométriques inverses, voir la rubrique ARCCOS.

```
  5 PRINT
 10 PRINT "LA FONCTION ARCSINUS
"
 20 PRINT
 25 PRINT
 30 FOR I=-1 TO 1 STEP .25
 40 PRINT "ARCSIN(";I;")";TAB 1
5;" = "; ASN I
 50 NEXT I
```

Figure A.10 : ARCSIN-Exemple de programme

```
LA FONCTION ARCSINUS

ARCSIN(-1)          = -1.5707963
ARCSIN(-Ø.75)       = -Ø.848Ø62Ø8
ARCSIN(-Ø.5)        = -Ø.52359878
ARCSIN(-Ø.25)       = -Ø.25268Ø26
ARCSIN(Ø)           = Ø
ARCSIN(Ø.25)        = Ø.25268Ø26
ARCSIN(Ø.5)         = Ø.52359878
ARCSIN(Ø.75)        = Ø.848Ø62Ø8
ARCSIN(1)           = 1.57Ø7963
```

Figure A.11 : ARCSIN-Exemple de résultat

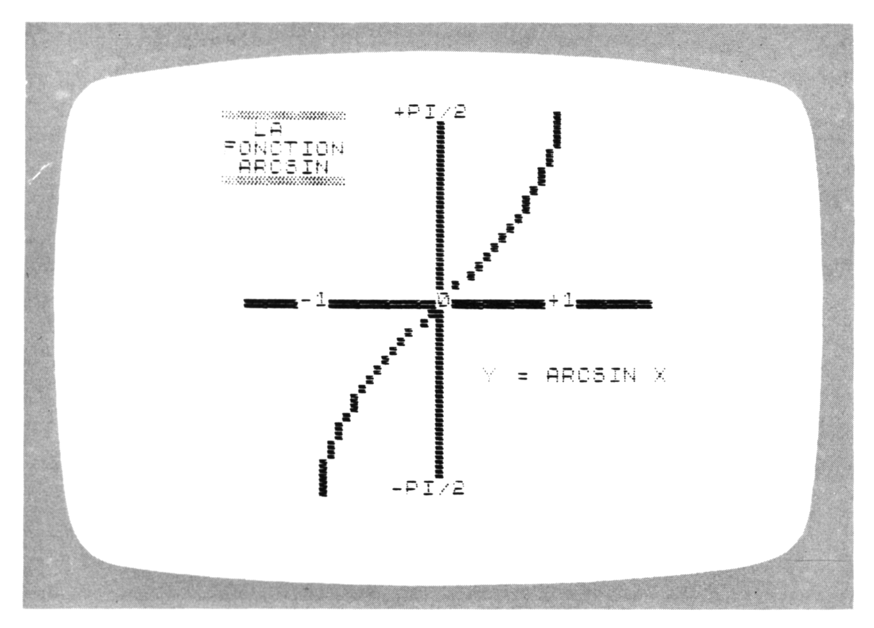


Figure A.12 : ARCSIN-Tracé de courbe

ARCTAN (fonction ; touche [D])

La fonction ARCTAN donne l'arctangente de n'importe quel argument négatif ou positif. (L'arctangente d'un nombre x est par définition l'angle dont la tangente est x.) Le résultat de la fonction ARCTAN s'exprime en radians.

Exemple de programme

Le programme de la Figure A.13 illustre la fonction ARCTAN. Elle est représentée sur l'écran par ATN comme on le voit ligne 40. La Figure A.14 montre le résultat de ce programme. L'arctangente de 0 est 0. Le résultat de la fonction ARCTAN tend vers $+\pi/2$ quand l'argument tend vers $+\infty$; inversement, le résultat de la fonction ARCTAN tend vers $-\pi/2$ quand l'argument tend vers $-\infty$.

```
   5 PRINT
  10 PRINT "LA FONCTION ARCTANGE
NTE"
  20 PRINT
  30 FOR I=-8 TO 8
  40 PRINT "ARCTAN(";I;")";TAB 1
3;"= ";  ATN I
  50 NEXT I
```

Figure A.13 : ARCTAN-Exemple de programme

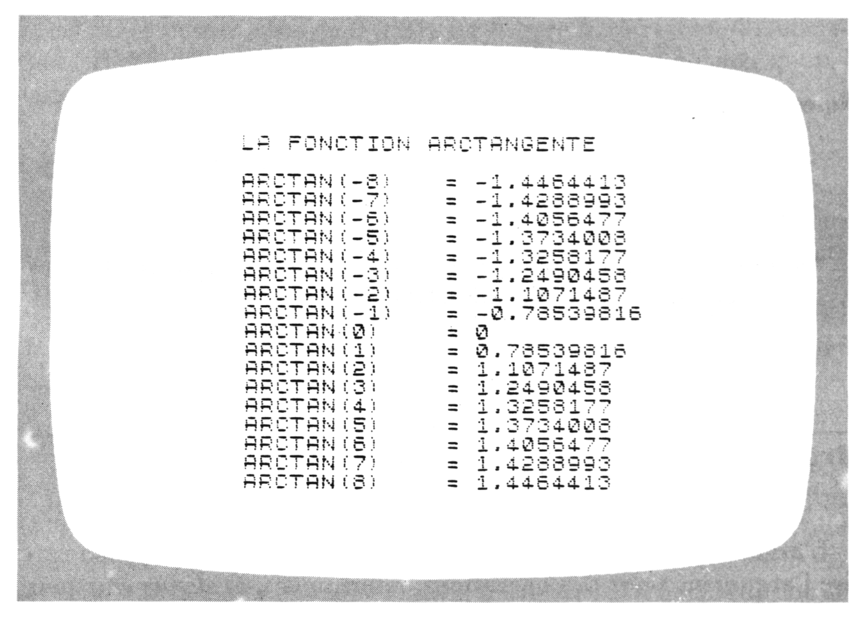


Figure A.14 : ARCTAN-Exemple de résultat

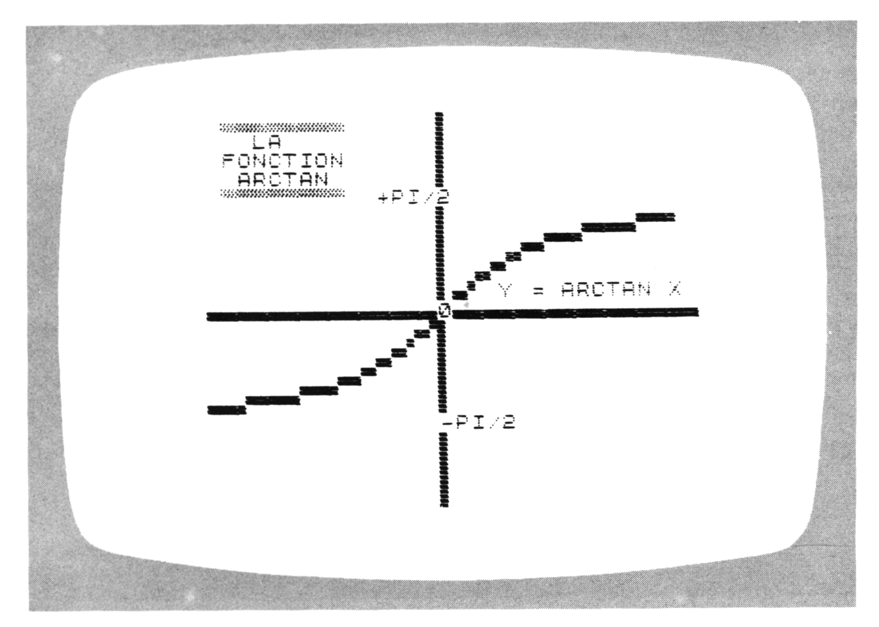


Figure A.15 : ARCTAN-Tracé de courbe

Notes et commentaires

- La Figure A.15 montre un tracé sommaire de la branche principale de la fonction ARCTAN.

- Pour plus d'information sur les fonctions trigonométriques inverses, voir la rubrique ARCCOS.

Argument (vocabulaire informatique)

L'argument est la valeur associée à une fonction. La fonction utilise l'argument pour des opérations déterminées, et *donne en retour* une autre valeur. Il faut remarquer que certaines fonctions n'ont pas besoin d'arguments.

AT (fonction ; touche [D])

La fonction AT est utilisée uniquement avec l'instruction PRINT. Elle détermine sur l'écran l'endroit où l'instruction PRINT inscrira une information. La fonction AT doit toujours être suivie par une adresse à double coordonnée de la forme suivante :

PRINT AT R,C ; "INFORMATION"

Les coordonnées R et C de l'adresse peuvent être des valeurs numériques littérales, des variables ou des expressions arithmétiques. La valeur absolue de chacune des coordonnées ne doit pas sortir des limites suivantes :

0< =R< =21
0< =C< =31

L'ordinateur utilise ces coordonnées pour identifier une position sur une grille imaginaire quadrillant l'écran. Cette grille est constituée de 22 rangées (horizontales) numérotées de 0 à 21 et de 32 colonnes numérotées de 0 à 31. La Figure A.16 résume l'instruction PRINT AT, et indique les adresses de la fonction AT aux quatre coins de l'écran.

Exemple de programme

Le petit programme de la Figure A.17 montre deux utilisations différentes de l'instruction PRINT AT et en particulier les différents types de coordonnées qui lui sont associées.

La ligne 10 du programme se contente d'imprimer un titre à partir de la position (1,7). Dans ce cas, les coordonnées d'adresse sont exprimées en valeurs numériques littérales :

PRINT AT 1,7 ; "LA FONCTION ""AT"

Cette instruction permet l'impression de la première lettre du titre —"L"— à la position (1,7), et de la suite du titre sur la même ligne. (Il ne faut pas oublier que la position (1,7) se trouve à l'intersection de la deuxième rangée et de la huitième colonne, étant donné que les numéros des rangées et des colonnes commencent à 0.)

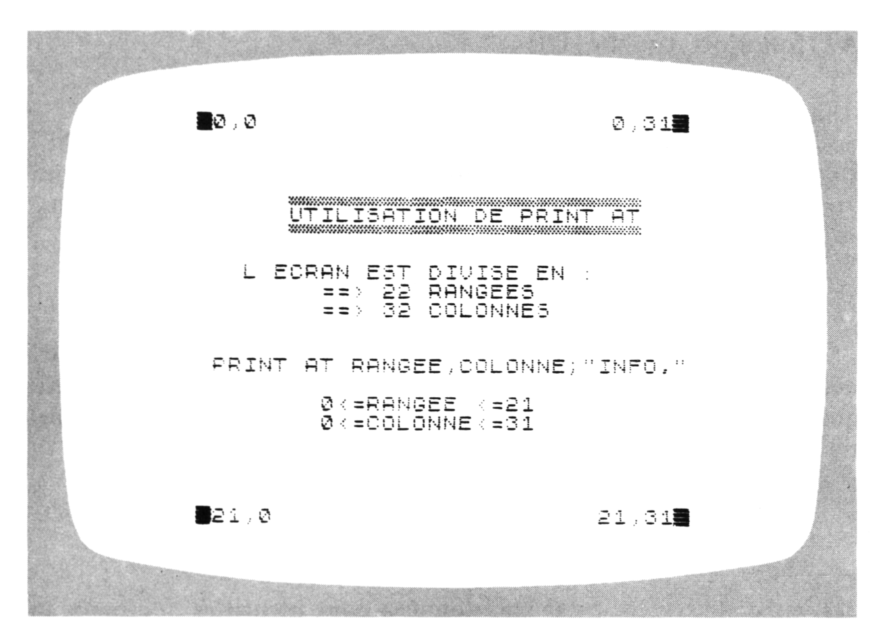


Figure A.16 : PRINT AT-Résumé

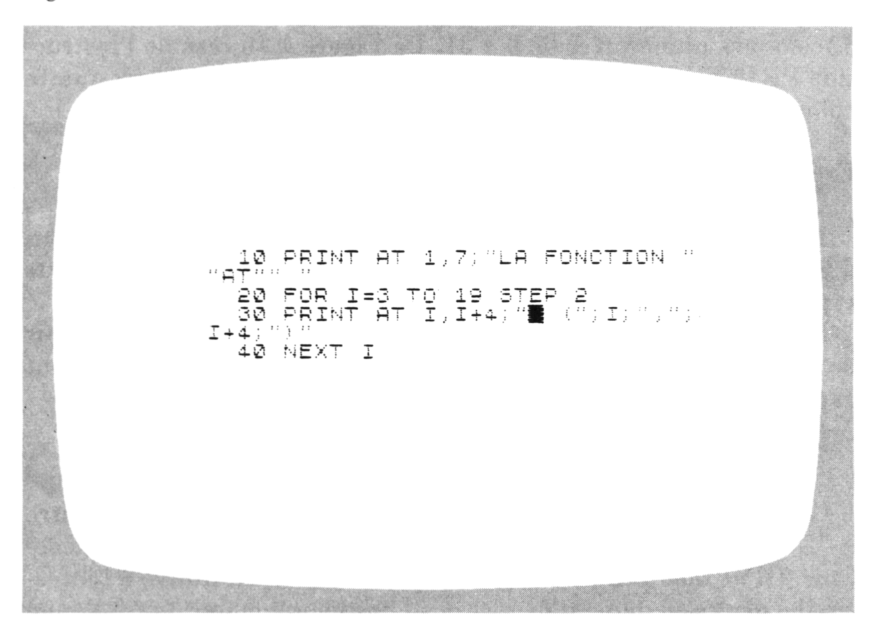


Figure A.17 : PRINT AT-Exemple de programme

La ligne 30 du programme montre comment les coordonnées peuvent également être exprimées sous forme de variables ou d'expressions arithmétiques :

30 PRINT AT I,I+4;...

Dans ce cas l'abscisse correspond à la variable I et l'ordonnée à l'expression I + 4. La variable I est en fait l'indice d'une boucle FOR qui commence à la ligne 20. On voit en étudiant le résultat de ce programme (Figure A.18) que la ligne 30 provoque l'impression de petits carrés noirs avec l'adresse de leur position. (Pour obtenir un carré noir, il faut d'abord passer en mode graphique — SHIFT-9 — puis appuyer sur la touche d'espacement (SPACE).

Notes et commentaires

- Une instruction PRINT AT dont les coordonnées d'adresse sont en dehors des limites permises entraîne un ou deux messages d'erreur :

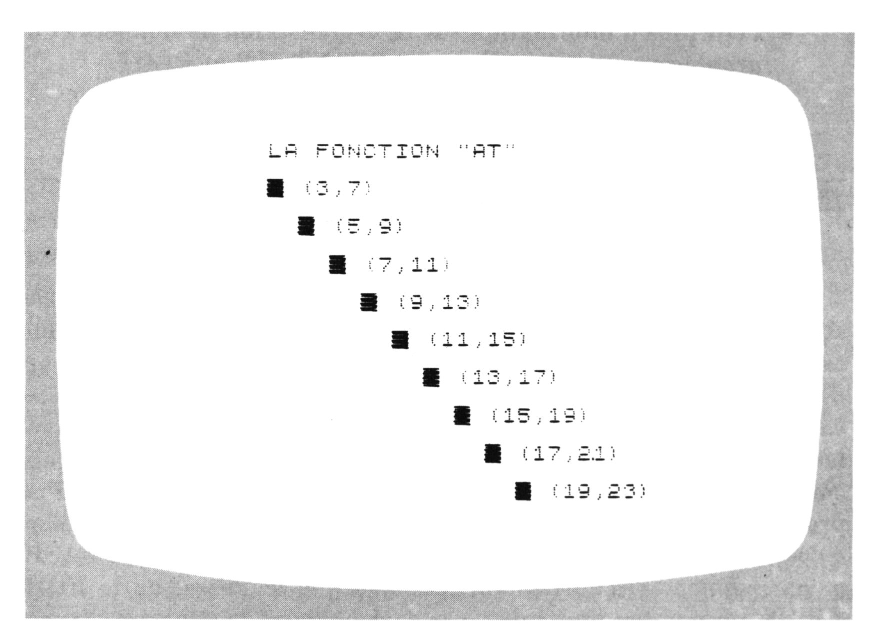


Figure A.18 : PRINT AT-Exemple de résultat

1. Un message d'erreur "5", si l'abscisse est 22 ou 23. (Ces deux rangées sont un espace de travail réservé par l'ordinateur, et ne sont donc pas disponibles pour l'impression d'informations.)
2. Un message d'erreur "B", si n'importe quelle coordonnée dépasse les limites autorisées.

- Les coordonnées d'adresse avec des valeurs négatives ou des valeurs non-entières sont permises. L'ordinateur convertit les coordonnées négatives en leurs valeurs absolues ; par exemple, l'adresse (-5,-7) deviendra (5,7). Les valeurs décimales sont arrondies au nombre entier supérieur ou inférieur le plus proche ; l'adresse (7.4,6.5) deviendra (7,7).

- Une instruction PRINT peut contenir de multiples expressions AT ; par exemple :

```
PRINT AT 3,3;"EN-HAUT A GAUCHE"; AT 20,20;
"EN-BAS A DROITE"
```

- Pour plus d'information sur la visualisation de données et de graphiques, voir les rubriques PRINT, AB et PLOT.

BASIC (vocabulaire informatique)

Le sigle BASIC (*Beginner's All-purpose Symbolic Instruction Code*) signifie : "code d'instructions symboliques universelles pour débutant". C'est un langage de programmation disponible sur la plupart des microordinateurs actuellement sur le marché. Le BASIC se caractérise par un jeu d'instructions puissant et par la facilité avec laquelle on l'utilise. Cependant la pléthore de *versions* de ce langage complique la situation, et rend parfois difficile le transfert d'un programme BASIC d'un microordinateur à un autre. Chaque version a des caractéristiques et des capacités qui lui sont propres. Le BASIC a, par rapport à d'autres langages tels que le Pascal ou le FORTRAN, un nombre limité de *types de données* et de *structures de contrôle itératives*. Le BASIC du ZX 81 possède seulement deux types de

données (données numériques et chaînes de caractères) et deux façons de créer des boucles itératives (les instructions FOR et GOTO). Il faut remarquer également que toutes les variables du BASIC sont des *variables globales*, c'est-à-dire que toutes les variables définies dans un programme sont utilisables à n'importe quel endroit de celui-ci. Il est difficile de créer des variables locales pour un sous-programme ou de transférer des valeurs d'un sous-programme à un autre.

BREAK (touche [SPACE])

Si l'on appuie sur la touche BREAK lorsque l'ordinateur exécute un programme, celui-ci est interrompu. Le mot BREAK ne fait pas réellement partie du vocabulaire BASIC du ZX 81 ; on ne peut pas inclure BREAK dans une instruction de programme. La touche BREAK permet seulement d'interrompre le programme en cours avant la fin de son exécution.

Exemple de programme

Les instructions du programme de la Figure B.1 forment une boucle sans fin, c'est-à-dire une séquence d'instructions que l'ordinateur exécute indéfiniment s'il n'est pas interrompu. Le rôle du programme est de remplir l'écran en répétant une phrase unique (lignes 10 à 30), d'effacer l'écran (ligne 40) puis de reprendre l'ensemble du processus (ligne 50).

Une fois que le programme tourne, on peut l'interrompre en appuyant sur la touche BREAK. On verra alors apparaître quelque chose de semblable à ce qui est sur l'écran de la Figure B.2. L'ordinateur a affiché dans le coin en bas à gauche le message suivant :

D/30

Cette lettre D est l'un des 16 messages d'erreur. Elle indique que le programme a été interrompu (par la touche BREAK dans le cas présent). Le nombre 30 indique que l'ordinateur était à la ligne 30 quand le programme a été interrompu.

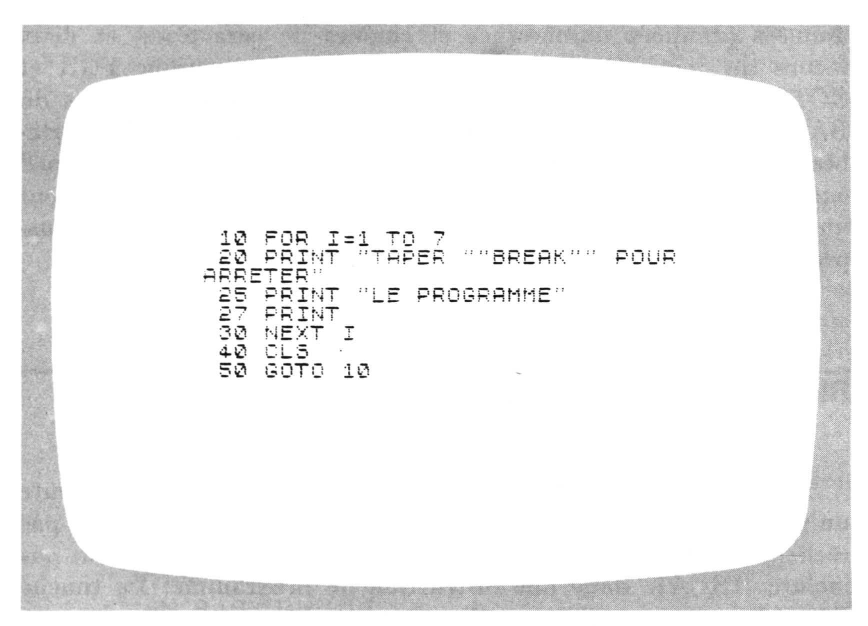


Figure B.1 : BREAK-Exemple de programme

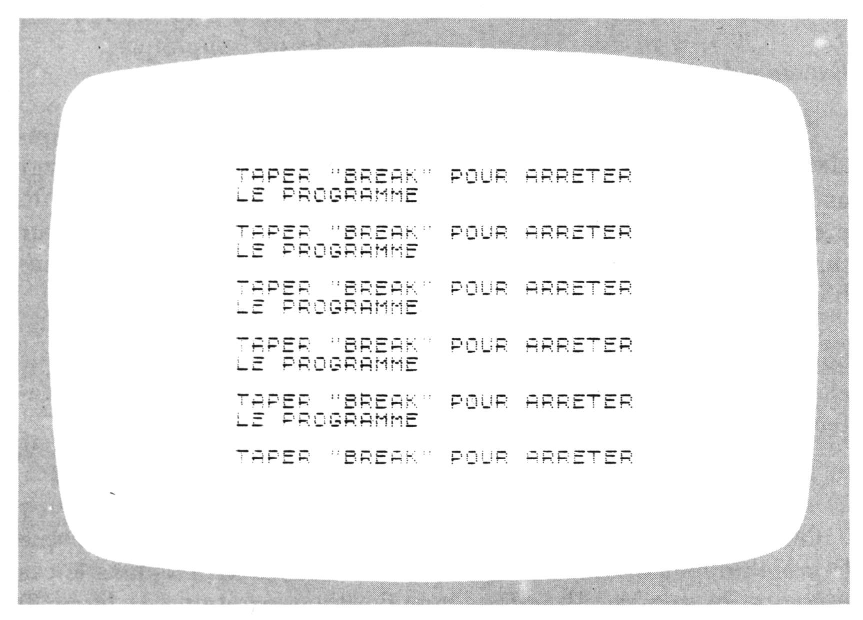


Figure B.2 : BREAK-Exemple de résultat

Notes et commentaires

- Si en cours d'exécution, l'ordinateur attend l'entrée d'une donnée en provenance du clavier (c'est-à-dire si l'ordinateur exécute une instruction INPUT), on ne peut pas utiliser la touche BREAK pour interrompre le programme. On doit utiliser alors l'instruction STOP pour arrêter celui-ci. Pour plus d'information, voir la rubrique STOP.

- On peut relancer le programme interrompu par la touche BREAK en utilisant la commande CONT. (Voir la rubrique CONT.)

Chaîne (vocabulaire informatique)

Une chaîne est un ensemble de données composé d'un ou de plusieurs caractères. L'ordinateur stocke chaque caractère dans un octet après l'avoir transcrit dans son équivalent en code numérique (voir les rubriques CHR$, CODE, STR$ et VAL). Le BASIC du ZX 81 offre, en plus des fonctions qui manipulent des arguments ou qui donnent des valeurs de chaînes, une méthode permettant d'accéder à des *sous-ensembles* de chaînes et de les manipuler. Cette méthode est appelée *découpage*. Elle utilise le mot clé TO pour identifier le sous-ensemble de la chaîne (voir rubrique TO).

CHR$ (fonction ; touche [U])

Le format de stockage des caractères du clavier du ZX 81 est un code numérique. Ce code contient 256 éléments compris entre 0 et 255. (Ce code ne correspond *pas du tout* au code ASCII qui est le code le plus couramment utilisé sur les micro-ordinateurs.) Le code du ZX 81 comprend toutes les lettres, les chiffres, les signes de ponctua-

tion, les caractères graphiques (y compris les caractères en vidéo inverse), ainsi que les mots clés BASIC et les noms de fonctions qui apparaissent sur le clavier. Cette combinaison a pour conséquence de permettre le stockage, sous une forme codée, de tout caractère ou de tout mot du vocabulaire BASIC dans un seul octet de la mémoire de l'ordinateur.

La fonction CHR$ donne le caractère correspondant à un code donné. L'argument de CHR$ doit être un nombre compris entre 0 et 255 ; le résultat est un caractère ou mot clé BASIC qui correspond à ce nombre.

Exemple de programme

Le programme de la Figure C.1 illustre l'emploi de la fonction CHR$ et permet d'examiner un échantillon du code caractère du ZX 81. Le programme invite l'utilisateur à introduire le code du premier caractère de la série à examiner ; il affiche ensuite sur l'écran 18 codes et les caractères qui leur correspondent.

```
  10 PRINT AT 18,0;"POUR EXAMINE
R UNE PARTIE DU "
  20 PRINT AT 19,0;"CODE CARACTE
RE DU ZX 81,"
  30 PRINT AT 20,0;"TAPER UN NOM
BRE COMPRIS ENTRE : "
  40 PRINT AT 21,0;"  0 ET 238."
  50 INPUT N
  60 IF N>238 THEN GOTO 50
  70 CLS
  80 PRINT "  CODE","CARACTERE"
  90 PRINT
 100 FOR I=N TO N+17
 110 PRINT "    ";I;"    ";CHR$ I
 120 NEXT I
 130 PRINT
 140 PRINT "TAPER ""ENTER"" POUR
 CONTINUER"
 150 INPUT A$
 160 CLS
 170 GOTO 10
```

Figure C.1 : CHR$-Exemple de programme

Les lignes 100 à 120 forment une boucle qui affiche les codes. L'index I de la boucle FOR devient le code et l'argument de CHR$:

```
110 PRINT "";I,"";CHR$ I
```

La Figure C.2 montre un exemple de résultat affichant les codes 7 à 24.

Notes et commentaires

- Le tableau de la Figure C.3 montre la place des codes de plusieurs catégories de caractères. (La Figure C.1 permet d'examiner plus en détail chaque catégorie.)
- Si l'argument de CHR$ est en dehors des limites autorisées, (0 à 255), un message d'erreur « B » apparaît.
- Pour plus d'information, voir la rubrique CODE.

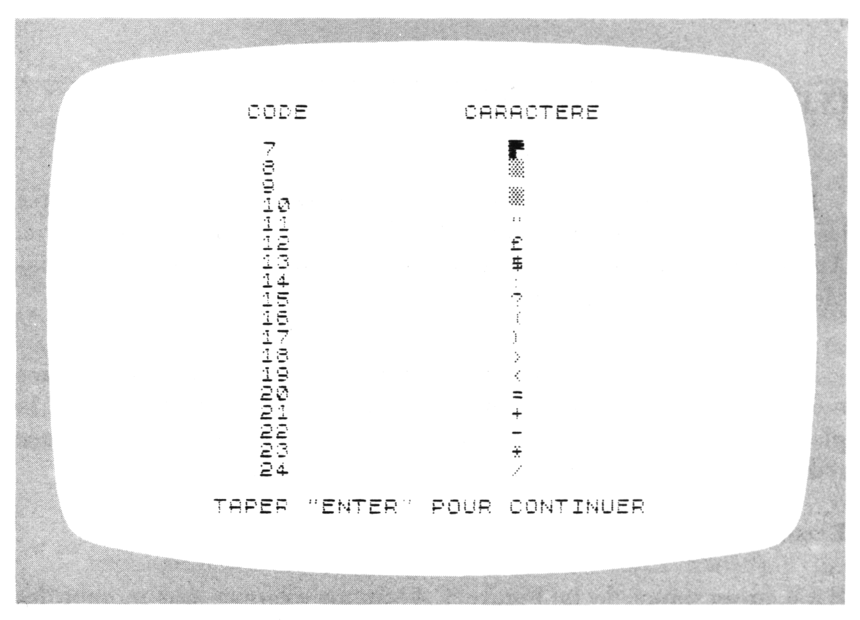


Figure C.2 : CHR$-Exemple de résultat

CODE-CARACTERE DU ZX 81

0 - 10	CARACTERES GRAPHIQUES
11 - 27	PONCTUATION ET SYMBOLES D OPERATIONS
28 - 37	CHIFFRES, 0 - 9
38 - 63	LETTRES, A - Z
67 - 127	INUTILISE
128 - 191	CARACTERES VIDEO-INVERSE
64 - 66 ET 192 - 255	MOTS-CLEF BASIC NOMS DE FONCT.

Figure C.3 : CHR$-Table des codes caractères

CLEAR (mot clé ; touche [X])

L'instruction CLEAR efface de la mémoire de l'ordinateur toutes les variables définies et en cours d'utilisation ainsi que leurs valeurs. Après l'exécution de cette instruction, on doit redéfinir toute variable nécessaire au programme.

Etant donné que les variables et leurs valeurs occupent de l'espace mémoire, l'instruction CLEAR peut être utile pour libérer de la place en mémoire si elles ne sont plus utilisées ou si elles doivent être redéfinies ultérieurement au cours du programme.

Exemple de programme

Le programme de la Figure C.4 est un exercice qui montre les effets de l'utilisation de la fonction CLEAR. Le programme est

```
  10 PRINT "RESULTAT DE L INSTRU
CTION"
  15 PRINT "          """CLEAR""""
  20 PRINT " ------ -- --- -----
-- ---------"
  25 PRINT
  30 PRINT "> INITIALISATION DE
 X, Y, Z,"
  40 PRINT
  50 LET X=RND
  60 LET Y=RND
  70 LET Z=RND
  80 PRINT "   X = ";X
  90 PRINT "   Y = ";Y
 100 PRINT "   Z = ";Z
 105 PRINT
 110 PRINT "> EXECUTION DE ""CLE
AR"""
 120 CLEAR
 125 PRINT
 130 PRINT "> TENTATIVE D IMPRES
SION DES      VARIABLES: ";X;Y;Z
```

Figure C.4 : CLEAR-Exemple de programme

prévu pour montrer à l'utilisateur le déroulement des trois étapes de l'opération afin qu'il puisse en étudier les résultats au fur et à mesure. Dans la première étape, le programme attribue des valeurs aléatoires à chacune des trois variables : X, Y et Z. Ceci est réalisé dans les lignes 50 à 70 ; les trois lignes suivantes affichent sur l'écran les noms des trois variables ainsi que leurs valeurs. Dans la deuxième étape, ligne 120, la commande CLEAR est exécutée. Pour finir, dans la ligne 130 du programme, on essaye d'imprimer à l'écran les trois valeurs de X, Y et Z. La Figure C.5 montre le déroulement de ce programme. On étudiera chacune des trois étapes. Le programme est interrompu prématurément au cours de la troisième étape, et l'ordinateur affiche le message d'erreur suivant :

```
2/130
```

Ce message d'erreur "2"indique que l'on a essayé d'utiliser une variable non définie. Cela montre en fait que l'instruction CLEAR a effacé les variables X, Y et Z de la mémoire de l'ordinateur ; en

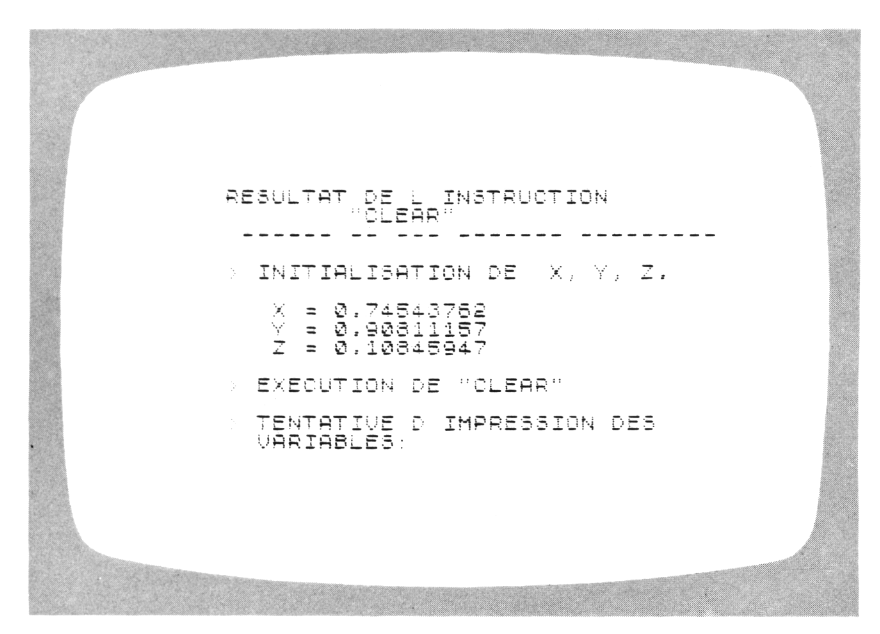


Figure C.5 : CLEAR-Exemple de résultat

conséquence, l'emploi de variables indéfinies, ligne 130, a provoqué l'arrêt prématuré du programme.

Notes et commentaires

- La commande RUN exécute une instruction CLEAR avant de commencer le déroulement du programme chargé en mémoire.

- Pour plus d'information sur la définition et l'initialisation des variables, voir les rubriques LET et INPUT.

CLS (mot clé ; touche [V])

Le mot clé CLS signifie "effacement d'écran". L'instruction CLS efface toutes les informations affichées à l'écran. Toute instruc-

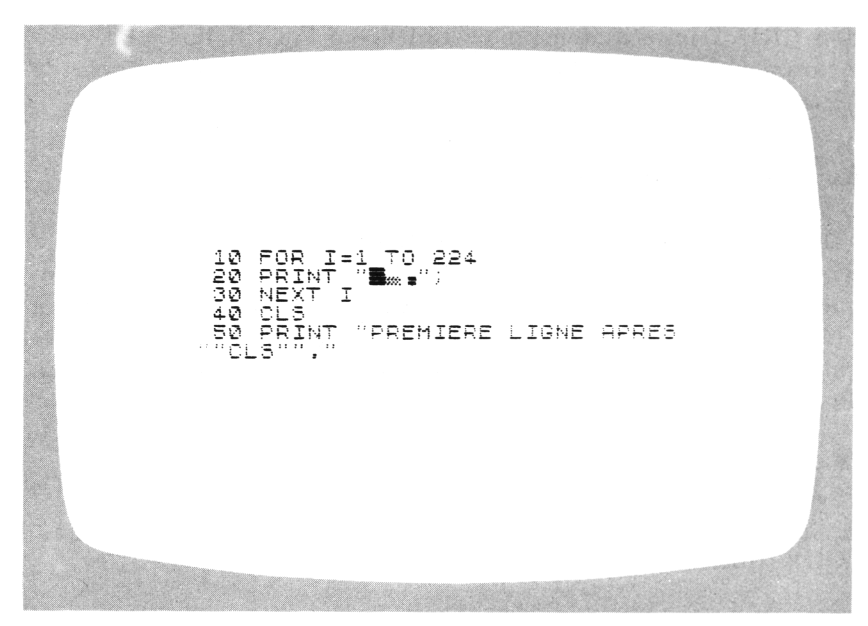


Figure C.6 : CLS-Exemple de programme

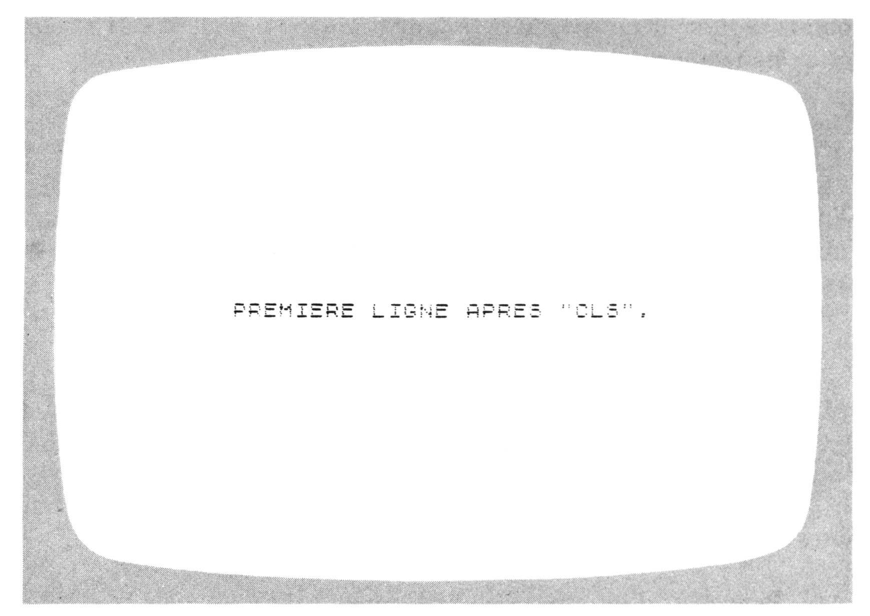


Figure C.7 : CLS-Exemple de résultat

tion PRINT immédiatement postérieure à l'instruction CLS commence à afficher des informations dans le coin supérieur gauche de l'écran.

Exemple de programme

Le petit programme de la Figure C.6 travaille en trois étapes : il remplit d'abord l'écran avec une configuration de caractères graphiques (lignes 10 à 30) ; il l'efface ensuite en exécutant l'instruction CLS (ligne 40) ; pour finir il imprime un message pour montrer ce qui suit l'instruction CLS. On voit Figure C.7 l'écran après exécution du programme.

CODE (fonction ; touche [I])

La fonction CODE est l'inverse de la fonction CHR$. Elle accepte comme argument un caractère unique et donne le code ZX 81 de ce caractère (compris entre 0 et 255).

Exemple de programme

Le programme de la Figure C.8 montre l'utilisation de la fonction CODE. Ligne 10 et 20, le programme lit un caractère du clavier par l'intermédiaire de la fonction INKEY$. La ligne 30 imprime le caractère ainsi que son code :

```
30 PRINT I$;" A POUR CODE "; CODE I$
```

Le programme forme une boucle sans fin (ligne 50) permettant l'examen d'un nombre illimité de codes. On voit Figure C.9 le déroulement du programme.

Notes et commentaires

- Pour plus d'information sur le code du ZX 81, voir la rubrique CHR$.

```
   5 PRINT "     CODE CARACTERE D
U ZX 81"
   7 PRINT
   8 PRINT
  10 LET I$=INKEY$
  20 IF I$="" THEN GOTO 10
  30 PRINT "       ";I$;" A POUR C
ODE : ";CODE I$
  40 PRINT
  50 GOTO 10
```

Figure C.8 : CODE-Exemple de programme

```
CODE CARACTERE DU ZX 81

 Z A POUR CODE : 63

 X A POUR CODE : 61

 8 A POUR CODE : 36

 1 A POUR CODE : 29

 B A POUR CODE : 39

 A A POUR CODE : 38

 S A POUR CODE : 56

 I A POUR CODE : 46

 C A POUR CODE : 40

 E A POUR CODE : 42
```

Figure C.9 : CODE-Exemple de résultat

Code machine (vocabulaire informatique)

Le code machine est l'ensemble des instructions codées en mémoire et directement exécutables par l'ordinateur. Pour écrire des programmes en code machine sur le ZX 81, il faut comprendre le jeu d'instructions et l'architecture du microprocesseur Z80. Le ZX 81 a des instructions qui permettent d'explorer le contenu des zones mémoire, de stocker des instructions en code machine dans des zones déterminées et "d'appeler" un sous-programme en code machine pendant l'exécution d'un programme BASIC. Voir les rubriques PEEK, POKE et USR.

Commande immédiate (vocabulaire informatique)

Une commande immédiate, par opposition à une commande insérée dans un programme, est une instruction que l'ordinateur exécute dès son entrée au clavier. A la différence des lignes d'instructions d'un programme BASIC, les commandes immédiates ne sont pas numérotées. De nombreuses instructions du BASIC ZX 81 peuvent être utilisées soit en commandes immédiates, soit en instructions de programme.

Concaténation (vocabulaire informatique)

La concaténation est le procédé de combinaison de deux chaînes pour en former une troisième. Le symbole plus (+) est utilisé, comme dans l'exemple suivant, pour représenter l'opération :

```
LET C$ = "CONCAT" + "ENATION"
```

Cette instruction LET a pour résultat le stockage de "CONCATENATION" dans la chaîne alphanumérique C$.

CONT (mot clé ; touche [C])

Le mot clé CONT signifie "continuer". Chaque fois que le déroulement d'un programme est interrompu prématurément — et quelle qu'en soit la cause — on peut utiliser la commande CONT pour que l'ordinateur reprenne le programme là où il l'a laissé. L'ordinateur garde en mémoire les valeurs de toutes les variables que le programme utilisait et la dernière ligne de programme exécutée. Après exécution de la commande CONT, ces valeurs sont toutes maintenues et le programme continue presque comme s'il n'avait jamais été interrompu. En fait, l'ordinateur efface l'écran avant de continuer.

Exemple de programme

Le programme de la Figure C.10 est destiné à montrer les capacités de l'ordinateur à utiliser la multiplication. Le programme est

```
  10 PRINT "UTILISATION DE ""CON
T"""
  20 PRINT
  30 FOR I=11  TO 20
  40 FOR J=11  TO 20
  50 PRINT I;" FOIS ";J;" EGAL "
;I*J
  60 NEXT J
  70 NEXT I
```

Figure C.10 : CONT-Exemple de programme

constitué de deux boucles FOR imbriquées (lignes 30 à 70). A chaque itération de la boucle interne, la ligne 50 imprime :

12 FOIS 13 EGAL 156.

Le programme est écrit pour aller de "11 FOIS 11" à "20 FOIS 20" ; cependant à chaque fois que l'écran est plein, l'exécution du programme est stoppée avec le message d'erreur :

5/50

Le message d'erreur "5" signifie simplement que le programme a rempli la totalité des 22 lignes de l'écran, et que l'ordinateur ne peut donc pas afficher la ligne suivante. C'est un problème courant en programmation, car souvent la capacité de l'écran ne permet pas au programme d'afficher la totalité des informations dont il dispose. Le ZX 81 possède une commande CONT pour résoudre ce problème. Quand l'examen de l'écran est terminé, il suffit d'entrer le mot clé CONT, ce qui entraîne :

1. effacement de l'écran,

2. reprise du programme là où il s'est arrêté.

Les résultats affichés par les deux écrans des Figures C.11 et C.12 illustrent ce processus (ces résultats sont obtenus après déroulement du programme de la Figure C.10). Le premier écran est plein à "12 fois 20" ; à ce stade, l'ordinateur stoppe le programme et affiche un message d'erreur :

5/50

Pour relancer le programme, il suffit de taper le mot clé CONT (comme le montre la Figure C.11) et de valider la commande. L'ordinateur poursuit le déroulement du programme (Figure C.12).

Notes et commentaires

- Si un programme est interrompu en cours d'exécution pour cause d'erreur (autre que la saturation de l'écran), l'ordina-

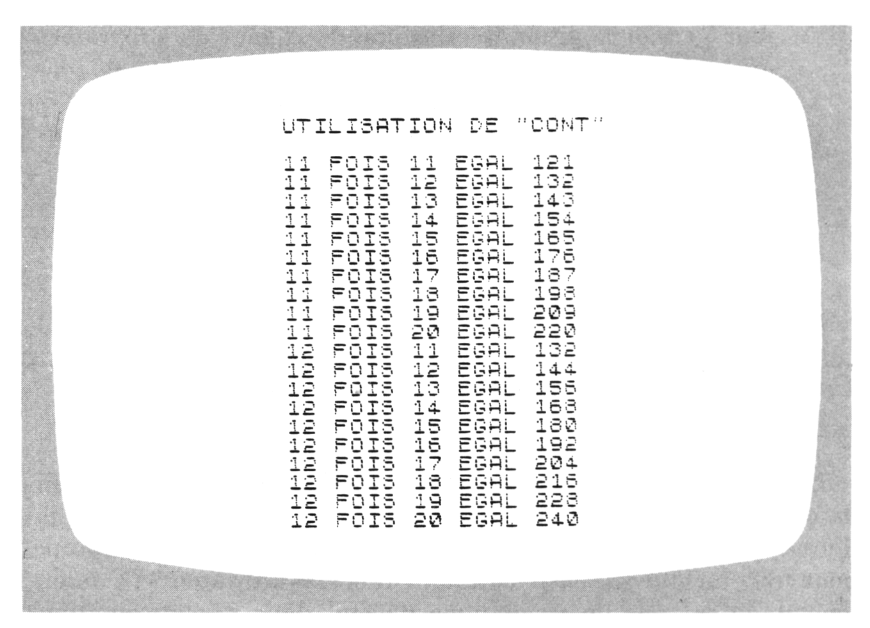


Figure C.11 : CONT-Exemple de résultat, premier écran

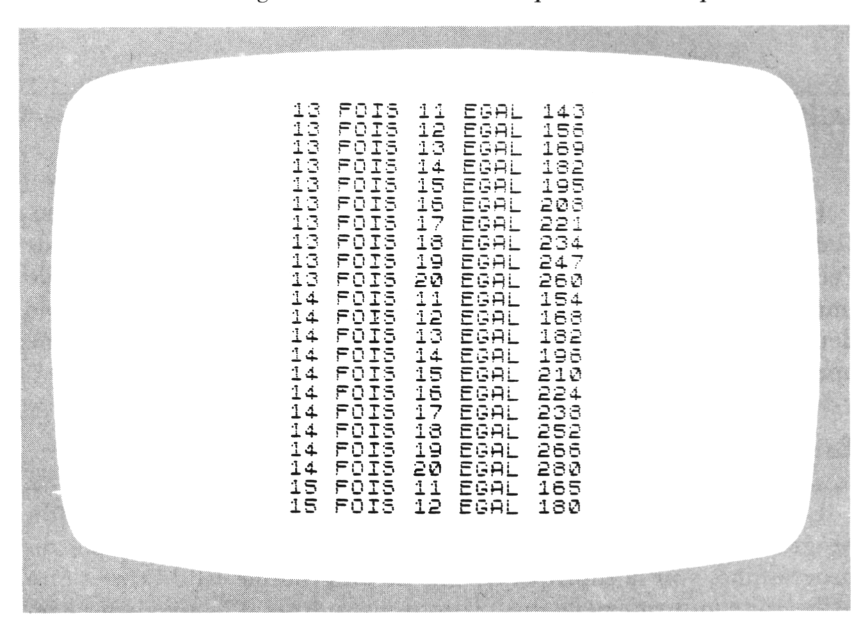


Figure C.12 : CONT-Exemple de résultat, second écran

teur permet l'édition de chacune des lignes du programme avant d'entrer la commande CONT. Après réception de l'instruction CONT, l'ordinateur reprend le déroulement du programme à partir du début de la ligne qu'il exécutait lors de l'interruption.

- Pour résoudre autrement le problème de la saturation d'écran, voir la rubrique SCROLL.

Conversationnel (vocabulaire informatique)

Le conversationnel est la capacité de l'ordinateur à répondre, pendant le déroulement d'un programme, aux informations que lui donne l'utilisateur. Un programme qui tire parti de cet avantage peut créer un *dialogue* entre l'ordinateur et son utilisateur. Le déroulement de ce type de programme dépend donc des informations entrées au clavier en cours d'exécution.

COPY (mot clé ; touche [Z])

Le mot clé COPY active l'imprimante du ZX 81, s'il y en a une. Il a pour effet d'envoyer sur l'imprimante tout ce qui est affiché sur l'écran. L'instruction COPY peut être utilisée soit en commande immédiate, soit en commande insérée dans un programme. Dans le deuxième cas, l'ordinateur continue le déroulement du programme après sortie sur imprimante du contenu de l'écran.

Exemple de programme

Le programme de la Figure C.13 est une reproduction du programme de la Figure A.3. (Pour une description complète du programme, voir la rubrique AND.) Le programme est destiné à être utilisé par un professeur qui traite des résultats d'examens à la fin d'un trimestre. L'adjonction de la commande COPY ligne 190 per-

```
  10 PRINT AT 21,0;"NOM ?"
  20 INPUT N$
  30 PRINT AT 20,0;"INTRODUCTION
 DES RESULTATS"
  35 LET QT=0
  40 FOR I=1 TO 3
  50 PRINT AT 21,0;"QUESTIONNAIR
E NO : ";I
  60 INPUT Q
  70 LET QT=QT+Q
  80 NEXT I
  90 LET MOY=QT/3
 100 PRINT AT 21,0;"EXAMEN FINAL
:"
 110 INPUT F
 120 CLS
 130 PRINT AT 3,3;N$
 150 PRINT " MOYENNE      = ";MOY
 160 PRINT " EXAMEN FINAL= ";F
 170 IF MOY>=75 AND F>=70 THEN P
RINT AT 9,5;"** SUCCES"
 180 IF MOY<75 OR F<70 THEN PRIN
T AT 9,5;"** ECHEC"
 190 COPY
 200 GOTO 10
```

Figure C.13 : COPY-Exemple de programme

met au professeur, une fois les données introduites, d'obtenir sur papier une copie d'écran des résultats de chaque élève.

Notes et commentaires

- Le ZX 81 possède deux autres commandes qui activent l'imprimante : LPRINT et LLIST. S'il n'y a pas d'imprimante reliée à l'ordinateur, ces trois commandes sont sans effet.

COS (fonction ; touche [W])

La fonction COS donne le cosinus d'un angle quelconque (négatif ou positif) exprimé en radians.

Exemple de programme

Le programme de la Figure C.14 affiche une série de cosinus pour des angles variant de 0 à 2 π. On voit le résultat de ce programme Figure C.15.

Notes et commentaires

- La Figure C.16 montre une courbe de la fonction cosinus de *x* pour des valeurs de *x* comprises entre 0 et 2 π. Cette courbe a été créée avec la commande PLOT.
- Etant donné que 180 degrés équivalent à π radians, on peut calculer qu'un degré équivaut approximativement à 0,0175 radians.
- La fonction π (située sous la touche [M]) est utilisée avec les fonctions trigonométriques (voir la rubrique π). Les autres fonctions trigonométriques disponibles en BASIC ZX 81 sont SIN et TAN.

```
   3 PRINT "  LA FONCTION COSINU
S "
   6 PRINT "  ==================
="
   9 PRINT
  10 FOR I=0 TO 8
  20 PRINT "COS(PI*";I/4;")","=
";COS (PI*I/4)
  30 NEXT I
```

Figure C.14 : COS-Exemple de programme

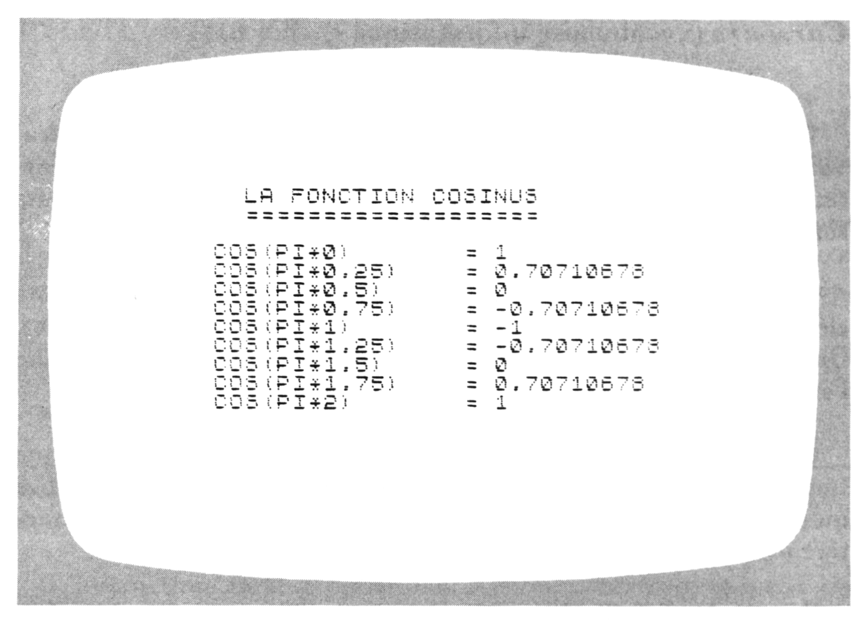


Figure C.15 : COS-Exemple de résultat

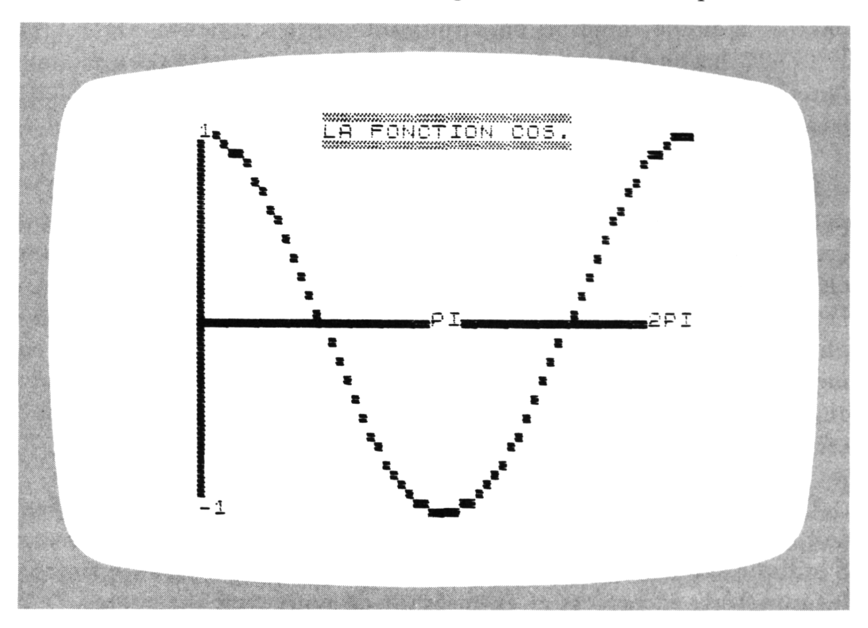


Figure C.16 : COS-Tracé de courbe

Curseurs (vocabulaire informatique — ZX 81)

Le ZX 81 possède cinq curseurs pour afficher, dans l'espace de travail de l'écran, le mode dans lequel se trouve le clavier. Chaque curseur est représenté par un caractère en vidéo inverse ; les curseurs sont les suivants :

Curseur-K : il indique que le clavier est en mode mot clé. Les mots clés sont inscrits au-dessus de chaque touche lettre sur le clavier. Quand l'ordinateur est en mode K, chaque touche tapée entraîne l'enregistrement sur la ligne d'édition du mot clé correspondant.

Curseur-L : il indique que le clavier est en mode lettre. Chaque touche tapée entraîne l'enregistrement de la lettre ou du chiffre qu'elle porte ou, si l'on appuie simultanément sur SHIFT, du caractère inscrit en rouge sur la touche.

Curseur-F : il indique que le clavier est en mode fonction. Les noms des fonctions sont inscrits au-dessous des touches. On peut passer en mode fonction en appuyant sur les touches SHIFT et ENTER. La touche que l'on tape ensuite est enregistrée comme une fonction ; l'ordinateur se replace ensuite automatiquement en mode lettre.

Curseur-G : il indique que le clavier est en mode graphique. On peut entrer en mode graphique n'importe lequel des 20 caractères graphiques qui sont inscrits sur le clavier (de 1 à 8 sur les touches chiffres, et sur les touches lettres Q, W, E, R, T, Y, A, S, D, F, G, H) ou entrer des caractères en vidéo inverse. Les 20 caractères graphiques doivent être tapés avec SHIFT. On peut mettre le clavier en mode graphique en tapant les touches SHIFT-9. L'ordinateur reste en mode graphique jusqu'à ce que l'on tape les touches SHIFT-9 une seconde fois.

Curseur-S : il indique une erreur de syntaxe soit dans les données, soit dans une ligne de programme que l'on vient d'entrer. Le curseur S indique la place de l'erreur. On peut corriger la ligne en supprimant les caractères incorrects et en insérant de nouveaux caractères.

DELETE (shift ; touche [0])

La touche DELETE a une fonction d'effacement que l'on peut utiliser soit en entrant une nouvelle ligne de programme dans l'ordinateur, soit en modifiant une ligne qui était stockée en mémoire. Une pression sur la touche DELETE efface le caractère, le mot clé ou le nom de la fonction située immédiatement à gauche du curseur (dans la zone de travail en bas de l'écran). Pour plus de renseignements, voir la rubrique EDIT.

DIM (mot clé ; touche [D])

L'instruction DIM ("dimension") permet au programmeur de définir un tableau et d'en spécifier les caractéristiques.

Une variable est un emplacement réservé dans la mémoire de l'ordinateur pour une certaine valeur; un tableau est un *ensemble* de variables qui sont indexées pour permettre un accès facile. Les tableaux sont dimensionnés. On parle de tableau à une dimension pour une *liste* de variables et de tableau à deux dimensions pour une *table* de variables.

D'autre part, chaque tableau a un nom spécifique, un genre et une longueur. Le mot clé DIM permet de définir facilement toutes ces caractéristiques en une instruction simple. Par exemple, l'instruction

```
DIM S(10)
```

définit un tableau numérique à une dimension appelé S et contenant 10 éléments. On sait que le tableau est unidimensionnel car les parenthèses suivant son nom ne contiennent qu'un nombre. Le nom lui-même spécifie le genre du tableau. Le nom d'un tableau numérique ne comporte qu'une lettre choisie de A à Z. Le nom d'un tableau alphanumérique est composé d'une lettre suivie du caractère "$".

On peut donc considérer le tableau S comme une liste de dix variables numériques. Les noms de ces dix variables sont :

```
S(1)
S(2)
S(3)
S(4)
S(5)
S(6)
S(7)
S(8)
S(9)
S(10)
```

Comme n'importe quelle variable numérique, chacune de ces variables ne peut stocker qu'une seule valeur à la fois.

Une fois le tableau S défini par l'instruction DIM, on peut utiliser ces 10 variables de la même façon que l'on emploie n'importe quelle variable numérique simple : on peut leur assigner des valeurs par l'intermédiaire des instructions LET ou INPUT. Il est possible d'afficher leurs valeurs à l'écran grâce à l'instruction PRINT ou d'intégrer ces variables à des expressions arithmétiques pour effectuer des calculs sur leurs valeurs.

On appelle *index* le nombre qui est entre parenthèses dans le nom du tableau. Celui-ci ne doit pas nécessairement être une valeur numérique littérale mais peut être représenté par une variable comme par exemple :

```
S(I)
```

Etant donné que la variable I a une valeur comprise entre 1 et 10 (longueur du tableau S), S(I) se rapporte à l'une de ces dix valeurs. On comprend pourquoi un tableau est un moyen pratique de stocker les données. En utilisant une variable comme index, on peut créer une relation entre un tableau et une boucle FOR pour réaliser, en utilisant très peu d'instructions, de longs travaux de traitement de données. Par exemple, les trois lignes suivantes pourraient afficher sur l'écran les dix valeurs stockées dans le tableau S :

```
100 FOR I=1 TO 10
110 PRINT S(I)
120 NEXT I
```

Au fur et à mesure que la boucle FOR incrémente la valeur de I de 1 à 10, on accède successivement à chaque valeur du tableau S et on les affiche sur l'écran. Cela implique bien sûr d'avoir assigné des valeurs à S auparavant dans le programme. On verra d'autres exemples de tableaux et de boucles FOR dans le programme ci-dessous.

L'instruction DIM peut, comme le tableau, être indexée par un nom de variable :

```
40 DIM S(N)
```

Dans ce cas, une valeur doit être assignée à la variable N *avant* que l'ordinateur ne rencontre l'instruction DIM lors du déroulement du programme. La valeur de N définira alors la longueur du tableau S.

Voici un exemple de définition de tableau à deux dimensions :

```
DIM T(3,4)
```

La *table* des variables représentée par le tableau T est :

```
T(1,1)   T(2,1)   T(3,1)
T(1,2)   T(2,2)   T(3,2)
T(1,3)   T(2,3)   T(3,3)
T(1,4)   T(2,4)   T(3,4)
```

La définition d'un tableau de chaîne de caractères est un peu plus compliquée que celle d'un tableau numérique. En BASIC ZX 81 chaque élément d'un tableau de chaîne de caractères a une longueur fixe et prédéterminée. Cela signifie que toutes les chaînes du tableau contiennent le même nombre de caractères. La longueur d'une chaîne doit être spécifiée dans l'instruction DIM qui définit le tableau. Par exemple, l'instruction

```
DIM N$(5,10)
```

définit le tableau alphanumérique unidimensionnel appelé N$. Ce tableau a cinq éléments :

```
N$(1)
N$(2)
N$(3)
N$(4)
N$(5)
```

Chacune de ces variables peut contenir une chaîne ayant exactement 10 caractères comme cela a été spécifié dans l'instruction DIM. Si l'on assigne à l'une de ces variables une chaîne dont le nombre de caractères est inférieur à 10, l'ordinateur complètera automatiquement par des espaces. Par exemple, l'instruction :

```
LET   N$(2)="BONJOUR"
```

est parfaitement équivalente à l'instruction :

```
LET   N$(2)="BONJOUR"
```

Inversement, si l'on essaie d'assigner à l'une de ces variables une chaîne dont le nombre de caractères est supérieur à 10, l'ordinateur tronquera les caractères excédentaires de façon à créer une chaîne de longueur correcte. Par exemple, l'instruction

```
LET N$(3)="ENCYCLOPEDIE"
```

stockera dans la variable N$(3) une chaîne alphanumérique de 10 caractères :

```
ENCYCLOPED
```

Exemple de programme

Le programme de la Figure D.1 est écrit pour un professeur qui veut comparer les résultats des épreuves d'un groupe d'élèves. Le programme définit deux tableaux : N$ pour stocker les noms des élèves, et S pour stocker leurs résultats. Etant donné que le programme peut être utilisé pour un nombre d'élèves variable, il faut disposer d'un moyen pour définir la *taille* des deux tableaux en interaction (c'est-à-dire pendant que le programme tourne). On remarque que les lignes 10 et 20 qui se trouvent juste avant les deux instructions DIM permettent au professeur d'introduire dans N une valeur correspondant au nombre d'élèves. Cette variable définit ensuite lignes 30 et 40 la longueur des deux tableaux :

```
30 DIM N$(N,10)
40 DIM S(N)
```

```
  10 PRINT AT 21,0;"NOMBRE D ETU
DIANTS : ?"
  20 INPUT N
  25 CLS
  30 DIM N$(N,10)
  40 DIM S(N)
  50 FOR I=1 TO N
  55 PRINT AT 20,0;"ETUDIANT :
";I
  60 PRINT AT 21,0;"NOM ? "
  70 INPUT N$(I)
  80 PRINT AT 21,0;"RESULTAT?"
  90 INPUT S(I)
 100 NEXT I
 110 CLS
 120 FOR I=1 TO N
 130 PRINT N$(I);TAB 10;":";
 135 PRINT S(I);TAB 15;
 140 FOR J=1 TO INT (S(I)/6+.5)
 150 PRINT "▒";
 160 NEXT J
 170 PRINT
 180 NEXT I
```

Figure D.1 : DIM-Exemple de programme

Les chaînes de caractères du tableau N$ auront toutes 10 caractères (si l'un des noms a plus de 10 lettres, il sera tronqué de façon à pouvoir entrer dans le tableau).

De la ligne 50 à la ligne 100, la boucle FOR est prévue pour exécuter l'entrée des données qui vont remplir les deux tableaux. Les lignes 55, 60 et 90 impriment sur l'écran des messages qui facilitent le processus d'entrée des données. Les instructions INPUT, lignes 70 et 90, utilisent la variable de contrôle I pour stocker les données entrées séquentiellement dans le tableau :

```
70 INPUT N$(I)
80 INPUT S(I)
```

Une fois les noms et les résultats des étudiants stockés dans les tableaux N$ et S, on peut réaliser rapidement et facilement autant de traitements de données que l'on veut. Les lignes 120 à 180 en donnent un bon exemple. Les boucles FOR imbriquées de ces lignes affichent sur l'écran les noms des élèves, leurs résultats et un

histogramme de ces résultats. Ce diagramme, dont on voit un exemple dans la Figure D.2, donne au professeur un outil visuel pour les comparer. Il ne faut pas oublier que *toutes* les données nécessaires à la réalisation de cet écran sont stockées dans les tableaux N$ et S.

Notes et commentaires

- La mise en application de l'instruction DIM en BASIC ZX 81 diffère de celle des autres langages sur deux points importants :

  1. Certaines versions du BASIC permettent de réaliser des tableaux courts (en général jusqu'à 10 éléments) utilisables dans un programme sans avoir été définis au préalable dans une instruction DIM. Ceci est *impossible* en BASIC ZX 81. Chaque tableau, quelle que soit sa longueur, nécessite une instruction DIM. De plus, chaque instruction DIM ne peut définir qu'un seul tableau. Cepen-



Figure D.2 : DIM-Exemple de résultat

dant, le nombre d'instructions DIM pouvant apparaître dans un programme est illimité.

2. Dans certaines versions du BASIC, la numérotation des éléments du tableau commence à 0 plutôt qu'à 1. Ainsi, par exemple, le tableau défini dans l'instruction suivante devrait contenir onze éléments :

```
DIM A(10)
```

Ces éléments seraient :

```
A(0)
A(1)
A(2)
...
A(10)
```

Ceci n'est *pas valable* pour le BASIC ZX 81, dans lequel les éléments du tableau sont indexés à partir du nombre 1.

- Une tentative d'accès à un élément dont l'indice est supérieure à celui du tableau entraîne un arrêt du programme suivi d'un message d'erreur "3". On prend par exemple le tableau E :

```
DIM E(10)
```

Il est impossible d'accéder à l'élément E(11).

EDIT (shift ; touche [1])

La touche EDIT, associée à plusieurs autres touches du clavier du ZX 81, permet de modifier n'importe quelle ligne d'un programme stocké dans la mémoire de l'ordinateur. Cette fonction de modification est extrêmement simple, d'une utilisation facile à apprendre et finalement très efficace.

Quand on appuie sur la touche EDIT (SHIFT [1]), l'ordinateur affiche automatiquement la ligne en cours dans la zone de travail située en bas de l'écran. On peut ensuite utiliser les touches de commande de déplacement du curseur vers la gauche ou vers la droite (SHIFT [5] et SHIFT [8] pour placer le curseur sur la partie de ligne à modifier. La touche d'effacement DELETE (SHIFT [0]) permet de supprimer des caractères, et les autres touches d'insérer de nouveaux caractères dans la ligne. Une fois les modifications terminées, on peut intégrer la ligne modifiée au programme par simple pression de la touche ENTER.

Exemple de programme

Imaginons que l'utilisateur ait entré dans l'ordinateur le programme de la Figure E.1. En regardant le listing affiché sur l'écran, il s'aperçoit qu'il a fait une erreur de frappe à la ligne 40 en écrivant :

(X+Y)**Z

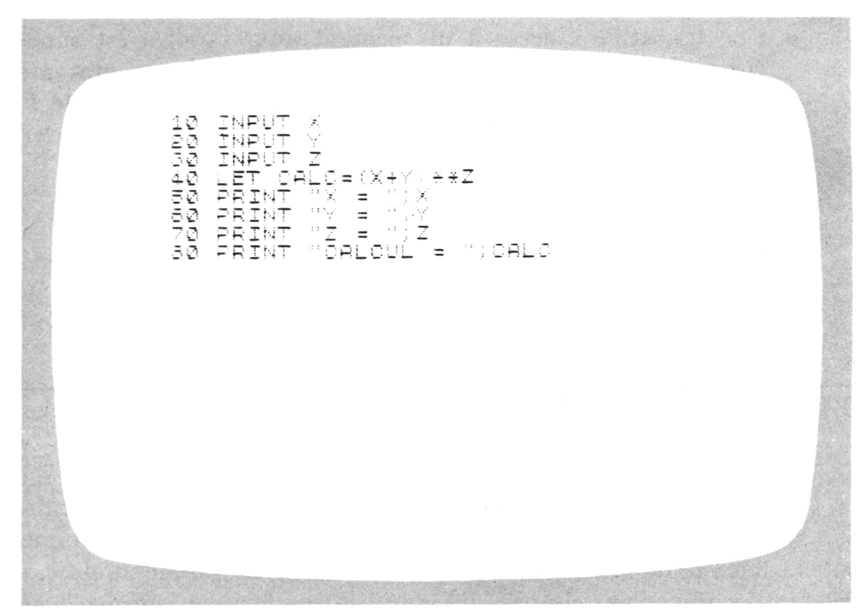


Figure E.1 : EDIT-Exemple de programme

au lieu de

```
(X-Y)**Z
```

Il pourrait bien sûr retaper toute la ligne, mais cela prendrait du temps et pourrait être une source d'erreurs supplémentaires. Il est plus facile d'utiliser la fonction EDIT. Les paragraphes suivants décrivent le processus de correction. Le moyen le meilleur et le plus rapide pour apprendre l'usage de la fonction d'édition est de l'utiliser de façon à en suivre avec attention les différentes étapes sur l'ordinateur.

La première étape est de désigner la ligne 40 comme ligne en cours. On remarque que la ligne 80 est signalée par un symbole « supérieur à » (>) en vidéo inverse. C'est le curseur de la ligne en cours (quand on écrit un programme, la ligne en cours est toujours la dernière ligne tapée). On peut utiliser trois méthodes différentes pour déplacer le curseur jusqu'à la ligne 40 :

1. Appuyer quatre fois sur la touche « flèche verticale » (SHIFT [7]). Le repère se déplacera d'une ligne vers le haut à chaque fois.

2. Entrer l'instruction :

   ```
   LIST 40
   ```

   Cette instruction affichera le programme à partir de la ligne 40 qui deviendra la ligne en cours.

3. Entrer un numéro de ligne inexistante compris entre 30 et 40, par exemple :

   ```
   39
   ```

   L'ordinateur réagira comme si l'on avait effacé une ligne numérotée 39 ; par conséquent, la ligne 40 deviendra ligne en cours.

Il serait bon d'utiliser ces trois méthodes de déplacement du curseur de ligne en cours. Chacune d'entre elles sera utile dans des situations différentes.

Lorsque la ligne 40 est devenue la ligne en cours, il suffit d'appuyer sur la touche EDIT (SHIFT [1]) pour déplacer cette ligne dans

```
10 INPUT X
20 INPUT Y
30 INPUT Z
40>LET CALC=(X+Y)**Z
50 PRINT "X = ";X
60 PRINT "Y = ";Y
70 PRINT "Z = ";Z
80 PRINT "CALCUL = ";CALC




40K LET CALC=(X+Y)**Z
```

Figure E.2 : EDIT-Programme de correction, effets de la touche EDIT

```
10 INPUT X
20 INPUT Y
30 INPUT Z
40>LET CALC=(X+Y)**Z
50 PRINT "X = ";X
60 PRINT "Y = ";Y
70 PRINT "Z = ";Z
80 PRINT "CALCUL = ";CALC




40 LET CALC=(X+LY)**Z
```

Figure E.3 : EDIT-Edition du programme : Déplacement du curseur EDIT

l'espace de travail de l'écran. Celui-ci doit alors ressembler à celui de la Figure E.2. On remarque que le curseur K se trouve sur cette ligne. Il devient le curseur d'édition.

La ligne est prête à être modifiée. Il faut appuyer plusieurs fois sur la touche « flèche à droite » jusqu'à ce que le curseur de modification se trouve juste à droite du caractère que l'on veut changer, dans le cas présent à droite du symbole "+". L'écran doit maintenant ressembler à celui de la Figure E.3. Pour finir, on peut appuyer sur la touche DELETE pour effacer le symbole "+", puis taper le signe moins (-) pour remplacer le caractère éliminé. Une fois la ligne modifiée, appuyez sur la touche ENTER, la nouvelle ligne 40 modifiée apparaît dans le programme.

Notes et commentaires

- Lors de la modification d'un très long programme, il peut être utile de mettre l'ordinateur en mode rapide (voir la rubrique FAST).

Espace de travail (vocabulaire informatique — ZX 81)

L'espace de travail est délimité par les lignes situées dans le bas de l'écran. Le ZX 81 les réserve pour son usage personnel. On ne peut pas accéder à cet espace avec une instruction PRINT. Lors de l'entrée d'informations sur le clavier, y compris lors de l'entrée de nouvelles lignes de programme, l'ordinateur affiche ce qui est tapé dans l'espace de travail. Quand on utilise la fonction EDIT, l'ordinateur affiche dans l'espace de travail une copie de la ligne que l'on veut éditer. Enfin, après chaque déroulement de programme, l'ordinateur affiche dans l'espace de travail un message de fin d'exécution ("message d'erreur"). Pour plus d'information sur l'espace de travail, voir les rubriques EDIT et Message d'erreur.

EXP (fonction ; touche [X])

La fonction EXP donne l'exponentielle naturelle d'un nombre (argument), c'est-à-dire le nombre *e* porté à la puissance de l'argument (où $e = 2{,}7182818$).

Exemple de programme

Le programme de la Figure E.4 est destiné à afficher une suite de valeurs de la fonction EXP, correspondant aux exponentielles d'une série d'arguments négatifs et positifs. On voit le résultat de ce programme Figure E.5.

Il faut remarquer que lorsque l'argument tend vers moins "l'infini", la valeur correspondante de l'exponentielle tend vers 0.

Notes et commentaires

- La Figure E.6 montre un tracé de courbe de la fonction EXP.

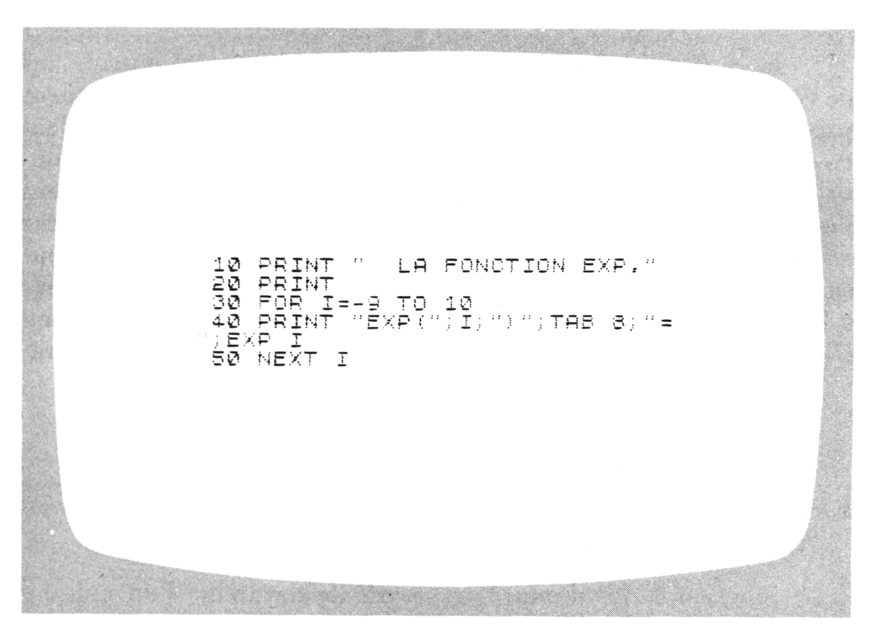


Figure E.4 : EXP-Exemple de programme

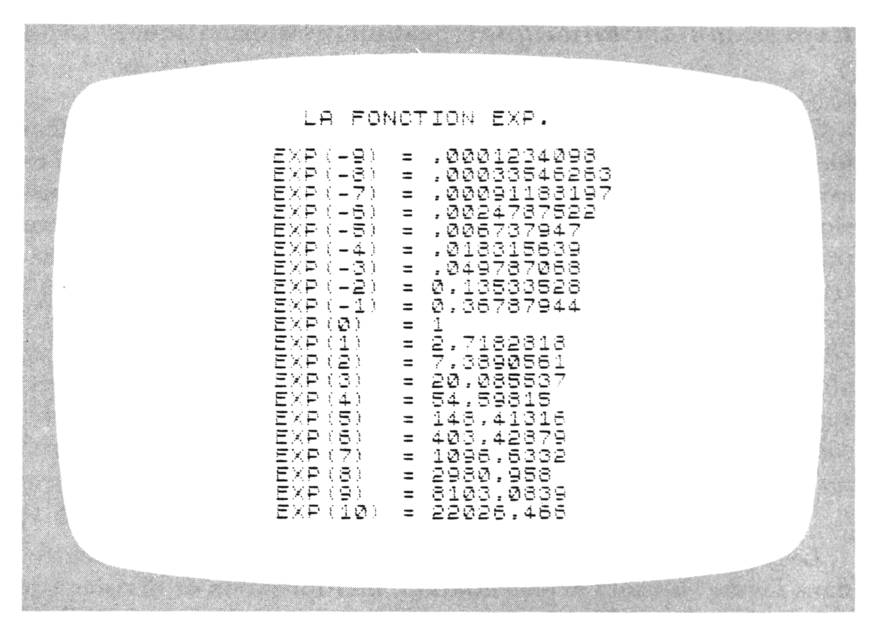


Figure E.5 : EXP-Exemple de résultat

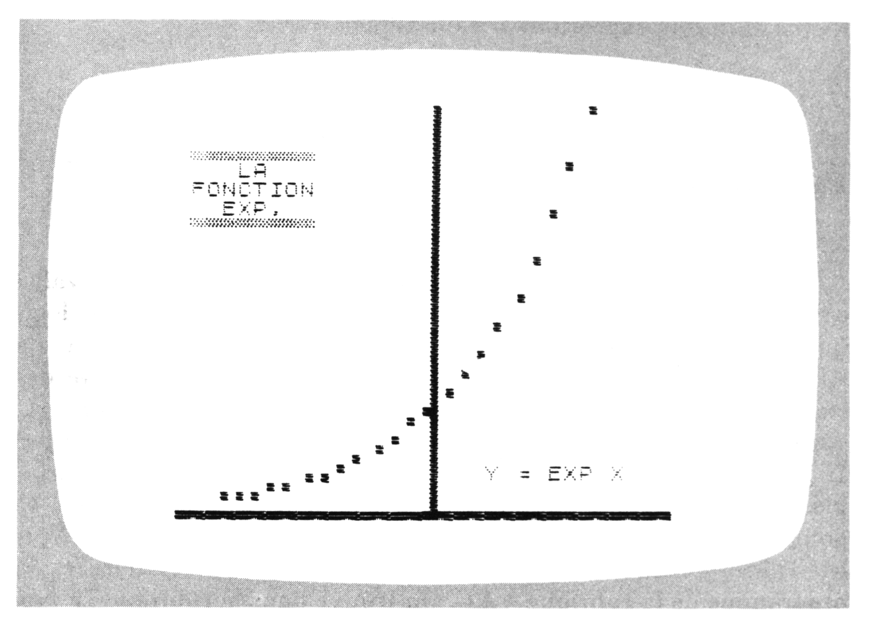


Figure E.6 : EXP-Tracé de courbe

Expression arithmétique (vocabulaire informatique)

Une expression arithmétique est une expression que l'ordinateur identifie comme une valeur numérique simple. Des valeurs numériques littérales, des noms de variables (représentant les valeurs numériques stockées dans ces variables), des fonctions et des opérations peuvent être des expressions arithmétiques. Les opérations arithmétiques sont représentées par les symboles suivants :

- ** exponentiation
- * multiplication
- / division
- + addition
- — soustraction

Les *opérations* sur les expressions arithmétiques sont exécutées dans l'*ordre* suivant : exponentiation, multiplication et division (de gauche à droite), addition et soustraction (de gauche à droite). Pour définir un ordre d'exécution différent, on peut introduire des parenthèses dans une expression arithmétique.

Expression logique (vocabulaire informatique)

Une expression logique est une expression que l'ordinateur évalue comme étant soit "vraie", soit "fausse". Les expressions logiques prennent toujours la forme d'égalités ou d'inégalités. Les mots AND (ET) et OR (OU) peuvent être utilisés pour construire des expressions logiques composées. La fonction logique NOT donne la négation de la valeur d'une expression logique. Dans une instruction IF, une expression logique peut être remplacée par une simple variable. Si la variable a pour valeur 0, elle sera estimée "fausse", et pour n'importe quelle autre valeur, elle sera estimée "vraie". Voir les rubriques IF, AND, OR et NOT ainsi que l'exemple de programme de la rubrique VAL. On trouve dans la rubrique IF une liste des symboles utilisés dans les expressions logiques.

FAST (shift ; touche [G])

La commande FAST met l'ordinateur en mode rapide. Quand il est dans ce mode, l'ordinateur arrête momentanément d'envoyer des informations à afficher sur l'écran et concentre son activité sur d'autres tâches. En conséquence, il les réalise plus rapidement.

On peut utiliser la commande FAST soit en mode immédiat, soit comme une instruction dans une ligne de programme. Quelle que soit l'utilisation choisie, l'ordinateur reste en mode rapide jusqu'à ce qu'on le remette en mode lent (voir la rubrique SLOW). Si l'ordinateur est en mode rapide pendant que l'on tape une ligne de programme, le clignotement produira un effet "haché" qui peut distraire ou gêner. D'un autre côté, pour gagner du temps, on peut mettre l'ordinateur en mode rapide lors de la modification d'un programme long.

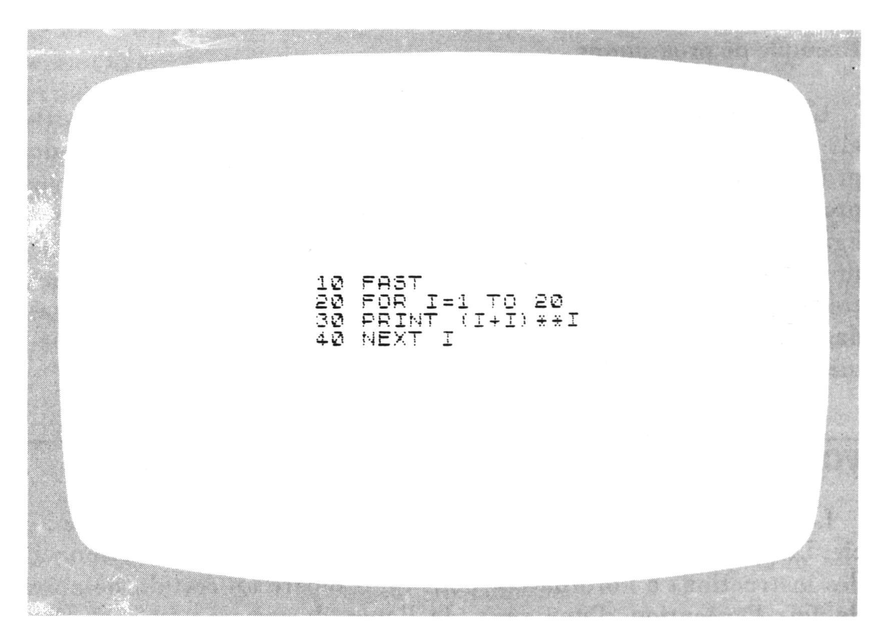


Figure F.1 : FAST-Exemple de programme

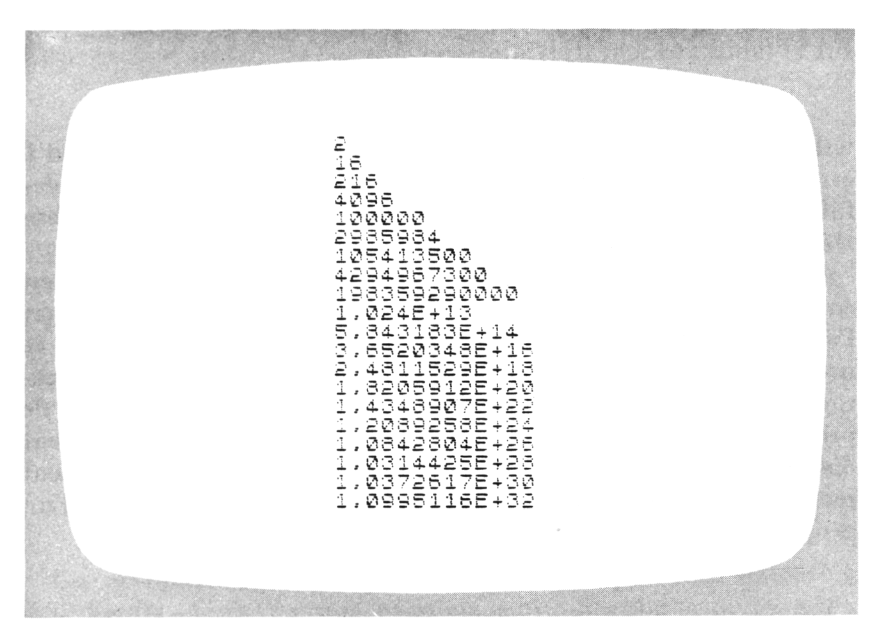


Figure F.2 : FAST-Exemple de résultat

Exemple de programme

Le programme de la Figure F.1 exécute une série de calculs et affiche les résultats sur l'écran. La Figure F.2 montre les résultats du programme. On peut comparer ces modes en faisant tourner le programme deux fois : la première fois en mode lent, sans la ligne 10 : la deuxième fois en mode rapide avec la ligne 10. Dans la première exécution, les lignes apparaissent sur l'écran une par une et très lentement. Dans la seconde exécution, l'écran devient gris pendant quelques secondes et la totalité des résultats apparaît brusquement.

FOR (mot clé ; touche [F])

Le mot clé FOR crée une structure itérative parfois appelée boucle. Le programmeur peut, en construisant une boucle FOR, donner des instructions à l'ordinateur pour qu'il répète un certain nombre de fois l'exécution d'une série de lignes de programme. Un des éléments essentiels de la boucle FOR est une variable de comptage

établie par l'ordinateur et appelée *variable de contrôle.* La valeur de cette variable change après chaque itération de la boucle, et l'ordinateur l'utilise pour déterminer le nombre de répétitions. Le programmeur peut aussi utiliser la valeur de cette variable de contrôle dans les instructions à répéter pour déterminer ce qui se passe à l'intérieur de la boucle. L'exemple de programme de cette rubrique en est l'illustration.

Deux autres mots BASIC, TO et NEXT, appartiennent à la syntaxe de la boucle FOR. TO indique la valeur que la variable de contrôle atteindra à la fin du processus d'itération. NEXT est le mot clé utilisé comme repère de la dernière ligne de la boucle. Dans l'exemple suivant :

```
 40 FOR I = 1 TO 10
          ...
100 NEXT I
```

la ligne 40 précise que I est la variable de contrôle de la boucle. On attribue initialement la valeur 1 à la variable I, qui sera *incrémentée* de 1 à 10 au cours du processus d'itération. La ligne 100 marque avec le mot NEXT la fin de la boucle FOR et se réfère à nouveau à la variable de contrôle I. Etant donné que cette variable prendra 10 valeurs différentes, les instructions situées entre la ligne 40 et la ligne 100 seront exécutées 10 fois, c'est-à-dire une fois pour chaque valeur de I.

N'importe quelle instruction BASIC peut être écrite à l'intérieur d'une boucle FOR. Les lignes à l'intérieur de la boucle utilisent souvent la variable de contrôle ; par exemple :

```
 40 FOR I=1 TO 10
 50 PRINT I
      ...
100 NEXT I
```

Lors du processus de répétition, la ligne 50 affiche sur l'écran les valeurs successives de la variable de contrôle.

Le BASIC permet également l'imbrication de boucles FOR les unes dans les autres ; on les appelle des boucles FOR *imbriquées* :

```
 40 FOR I=1 TO 10
 50 FOR J=1 TO 5
          ...
 90 NEXT J
100 NEXT I
```

Dans ce cas, à chaque répétition de la boucle externe, la boucle interne est répétée cinq fois. Les instructions des lignes 50 à 90 sont donc exécutées 50 fois en tout.

Pour construire deux boucles FOR imbriquées, il faut se souvenir de deux règles essentielles :

1. La boucle interne doit avoir sa propre variable de contrôle distincte de la variable de contrôle de la boucle externe (dans l'exemple ci-dessus, J est la variable de contrôle de la boucle interne).

2. La boucle externe doit englober la totalité de la boucle interne. C'est-à-dire que les lignes FOR et NEXT de la boucle interne doivent être entre les lignes FOR et NEXT de la boucle externe. La construction suivante est donc *interdite* :

```
10 FOR I=1 TO 10
20 FOR J=1 TO 5
         ...
50 NEXT I
60 NEXT J
```

La variable de contrôle d'une boucle FOR peut être exprimée sous différentes formes : valeurs numériques littérales (comme dans les exemples précédents), variables ou expressions arithmétiques, comme dans la ligne suivante :

```
40 FOR I=A TO B+C
```

Les variables A, B et C doivent être définies et initialisées avant la ligne 40. La variable de contrôle I sera incrémentée de 1, de A à B + C, au cours du processus d'itération.

L'utilisation du mot STEP permet, lors de chaque itération de la boucle, d'incrémenter la variable de contrôle d'une valeur différente de 1 ; par exemple :

FOR I=2 **TO** 10 **STEP** 2

Quand le mot STEP *n'apparaît pas* dans l'instruction FOR, la valeur *par défaut* de l'incrémentation est égale à 1. Pour plus d'information, voir la rubrique STEP.

Exemple de programme

On voit Figure F.3 un exercice classique qui montre les conséquences de l'utilisation de boucles FOR. Le programme affiche à l'écran une table de multiplication. Cette table est créée par les deux boucles imbriquées des lignes 10 à 70. La boucle interne (lignes 20 à 40) calcule et affiche une seule rangée de valeurs. La boucle externe répète cette opération sur les dix lignes de la table. La ligne 30 est

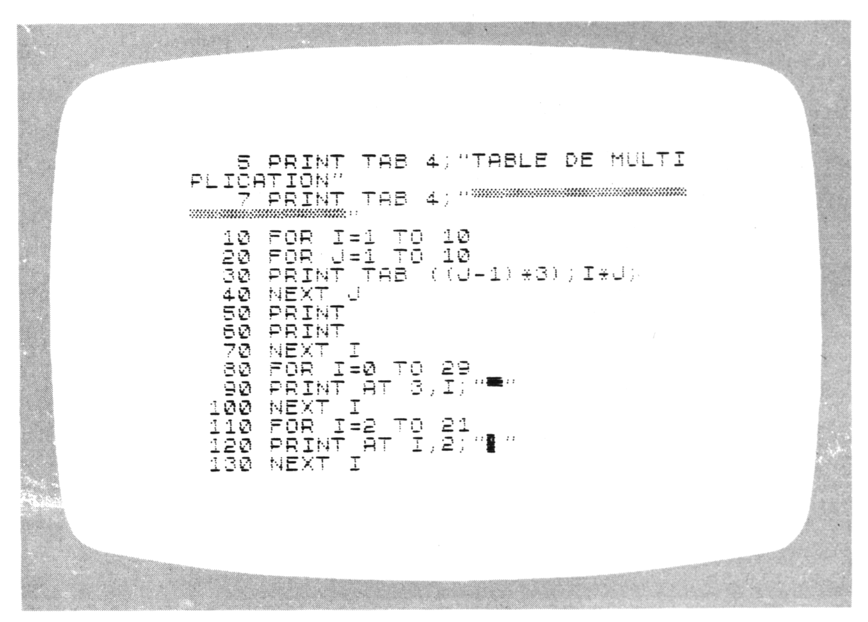


Figure F.3 : FOR-Exemple de programme

donc exécutée 100 fois, une fois pour chacune des 100 entrées de la table.

On remarque que la ligne 30 utilise les variables de contrôle I et J pour calculer chaque entrée :

```
I*J
```

et la variable de contrôle J pour faire une tabulation qui permette de placer correctement sur l'écran chaque nouvelle entrée :

```
TAB ((J-1) * 3)
```

Deux autres boucles FOR apparaissent dans le programme, lignes 80 à 100 et lignes 110 à 130. La première des deux crée une ligne horizontale et la seconde une ligne verticale qui délimitent respectivement la première rangée et la première colonne de la table. Les variables de contrôle de ces boucles deviennent (lignes 90 à 120) les coordonnées de l'instruction AT qui détermine l'emplacement de ces lignes.

La Figure F.4 montre le résultat de ce programme.

TABLE DE MULTIPLICATION

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Figure F.4 : FOR-Exemple de résultat

Notes et commentaires

- Si à la fin d'une boucle, on donne à la variable de contrôle de l'instruction NEXT le nom d'une variable qui existe réellement dans le programme, son déroulement sera interrompu par un message d'erreur "1". Les lignes suivantes illustrent cette erreur :

```
10 LET K=5
20 FOR I=1 TO 10
30 PRINT I*K
40 NEXT K
```

- A l'inverse des autres variables numériques du BASIC ZX 81, les noms des variables de contrôle ne doivent comporter qu'une seule lettre.

- Si la variable de contrôle, dont la gamme des valeurs a été définie dans l'instruction FOR, prend une valeur erronée, la boucle FOR n'aura aucun effet ; par exemple :

```
10 FOR I=1 TO 0
20 PRINT I
30 NEXT I
```

On remarque cependant que la boucle suivante a, grâce à l'instruction STEP, une variable de contrôle *décrémentée* ; elle affiche les valeurs de cette variable *allant* de 1 à 0 :

```
10 FOR I=1 TO 0 STEP -1
20 PRINT I
30 NEXT I
```

Pour plus d'informations sur les boucles FOR, voir la rubrique STEP.

Fonction (vocabulaire informatique)

Une fonction est un sous-programme immuable qui donne un certain type de valeur ou réalise une opération d'affichage sur l'écran. Le nom de la fonction appelle le sous-programme ; on obtient en retour l'expression arithmétique ou la chaîne alphanumérique qui lui correspond. La plupart des fonctions du ZX 81 nécessitent des *arguments*. Un argument est une valeur indispensable à la fonction. Mais certaines d'entre elles comme les fonctions RND et π n'ont pas besoin d'argument.

FONCTION (shift ; touche [ENTER])

La touche fonction place le clavier en mode fonction, ce qui permet d'entrer un nom de fonction sur une ligne de programme. Les noms des fonctions sont inscrits sur le clavier, *au-dessous* des touches lettres (excepté pour la lettre V). Le clavier revient automatiquement en mode lettre après chaque entrée de fonction.

GOSUB (mot clé ; touche [H])

L'instruction GOSUB dévie l'exécution du programme vers un sous-programme. Un sous-programme est une séquence d'instructions regroupées pour réaliser une tâche particulière. L'instruction GOSUB indique à l'ordinateur qu'il doit interrompre l'ordre séquentiel habituel (ligne par ligne) dans l'exécution du programme, pour se placer sur la première ligne d'un sous-programme défini, et commencer à en exécuter les instructions. L'instruction GOSUB implique obligatoirement la présence d'une instruction RETURN dans le sous-programme. Lorsqu'il rencontre cette instruction, l'ordinateur revient au programme (à la ligne suivant immédiatement l'instruc-

tion initiale GOSUB), et reprend l'exécution séquentielle du programme.

L'instruction GOSUB prend la forme suivante :

GOSUB L

dans laquelle L est le numéro de la première ligne du sous-programme. Ce numéro de ligne peut être exprimé de trois façons différentes en BASIC ZX 81 ; sous forme de valeur numérique littérale :

GOSUB 600

sous forme de variable, cette variable prenant la valeur du numéro de la première ligne d'un sous-programme :

GOSUB S

ou, enfin, sous la forme d'une expression arithmétique dont le résultat est un numéro de ligne de sous-programme :

GOSUB (S+1)*100

Quel que soit le mode d'expression du numéro de ligne, le résultat est identique : le contrôle de l'exécution du programme se déplace jusqu'à ce numéro de ligne et le programme se déroule séquentiellement jusqu'à la rencontre d'une instruction RETURN.

Du point de vue du programmeur, il y a plusieurs bonnes raisons de se donner le mal d'isoler certaines tâches de programmation dans des sous-programmes séparés. Tout d'abord, il y a, dans de nombreux programmes, des tâches qui doivent être exécutées plus d'une fois, à différents stades du programme. La répétition des lignes d'instructions nécessaires à la réalisation de ces tâches serait la cause d'une perte de temps pour le programmeur et d'un gaspillage de l'espace mémoire de l'ordinateur. La solution évidente à ce problème est d'écrire ces instructions une fois, de les isoler dans un sous-programme, et "d'appeler" ce sous-programme par l'intermédiaire d'une instruction GOSUB chaque fois que c'est nécessaire.

Le désir de réaliser des structures de programmes "modulaires" bien organisées est une raison supplémentaire d'utiliser des sous-

programmes. Des programmeurs expérimentés se sont rendu compte, souvent avec tristesse, qu'un programme long et complexe peut commencer, à un certain moment, à devenir "autonome" et terriblement difficile à contrôler, corriger ou modifier. Un moyen de combattre ce phénomène est de découper un long programme en séquences de tâches courtes et maniables qui, par une suite d'interactions, accomplissent l'ensemble du travail qui était destiné au programme. Dans cette répartition, on assigne à chaque sous-programme sa propre tâche et le programme principal se trouve réduit à une série d'appels de sous-programmes ou d'instructions GOSUB. Certains langages de programmation, comme par exemple le Pascal, sont spécialement conçus pour ce style de programmation modulaire "descendante". Bien qu'il manque au BASIC quelques-unes des caractéristiques essentielles qui rendent la programmation modulaire efficace, un programme BASIC peut souvent être considérablement amélioré si on le structure en petits sous-programmes soignés, faciles à comprendre et faciles à modifier.

Pour finir, plus on écrit de programmes, plus on écrit d'instructions pour des tâches similaires et même identiques. Si on prend l'habitude de créer des petits sous-programmes pour la réalisation des tâches courantes, on pourra les "transporter" mot pour mot, ou avec un minimum de modifications, dans d'autres programmes et simplifier ainsi le travail de programmation.

Exemple de programme

Les figures G.1 et G.2 montrent une charpente de programme "piloté" par menu et organisé suivant une structure modulaire descendante. Le programme ne *réalise* rien de concret, mais sert à illustrer d'une manière abstraite quelques principes de structuration de programmes. Il peut aussi servir de modèle pour l'écriture de programmes réels "pilotés" par menu.

Le premier écran (Figure G.1) représente ce que l'on pourrait appeler la partie "programme principal". Cette partie contrôle le déroulement du programme et a trois tâches principales à accomplir :

1. Afficher le "menu" sur l'écran

2. Recevoir le choix du menu de l'utilisateur

```
   3 REM *********************
   5 REM * PROGRAMME PRINCIPAL *
   7 REM *********************
   9 REM
  10 PRINT AT 5,5;"MENU"
  20 PRINT AT 6,5;"===="
  30 PRINT AT 8,3;"1) OPTION UN"
  40 PRINT AT 10,3;"2) OPTION DE
UX"
  50 PRINT AT 12,3;"3) OPTION TR
OIS"
  60 PRINT AT 14,3;"4) FIN"
  70 INPUT M$
  80 IF M$<"1" OR M$>"4" THEN GO
TO 70
  90 CLS
 100 GOSUB 700
 110 GOSUB (VAL M$+1)*100
 120 GOSUB 600
 130 CLS
 140 GOTO 10
```

Figure G.1 : GOSUB-Exemple de programme

```
 200 REM ** OPTION UN **
 210 PRINT TAB 11;"OPTION UN"
 220 RETURN
 300 REM ** OPTION DEUX **
 310 PRINT TAB 11;"OPTION DEUX"
 320 RETURN
 400 REM ** OPTION TROIS **
 410 PRINT TAB 11;"OPTION TROIS"
 420 RETURN
 500 REM ** FIN **
 510 PRINT TAB 11;"SALUT."
 520 GOSUB 700
 530 STOP
 600 REM ** CONTINUER **
 610 GOSUB 700
 620 PRINT AT 21,0;"CONTINUER?"
 630 INPUT A$
 640 RETURN
 700 REM ** LIGNE D ETOILES **
 710 PRINT "*******************
************"
 720 RETURN
```

Figure G.2 : GOSUB-Exemple de programme (suite)

3. Appeler le sous-programme approprié conformément au choix de l'utilisateur

Dans le vocabulaire informatique, le *menu* est un simple message destiné à expliquer à l'utilisateur quelles options sont disponibles à n'importe quel stade du déroulement du programme. Un menu devrait en plus lui indiquer clairement une façon simple d'exprimer son choix parmi les options. Ce menu, réalisé par les lignes 10 à 60, est affiché Figure G.3. Il propose quatre options : les trois premières sont appelées sans grande originalité UN, DEUX, TROIS et la quatrième FIN. Pour activer une de ces options, il suffit d'entrer le chiffre 1, 2, 3 ou 4 (la quatrième option permet à l'utilisateur de mettre fin au déroulement du programme).

On peut envisager avec un peu d'imagination de nombreuses situations dans lesquelles un menu identique à celui-ci serait un bon moyen d'offrir des choix à l'utilisateur. Par exemple, le programme pourrait être conçu pour un jeu quelconque, et le menu offrirait trois niveaux de difficulté :

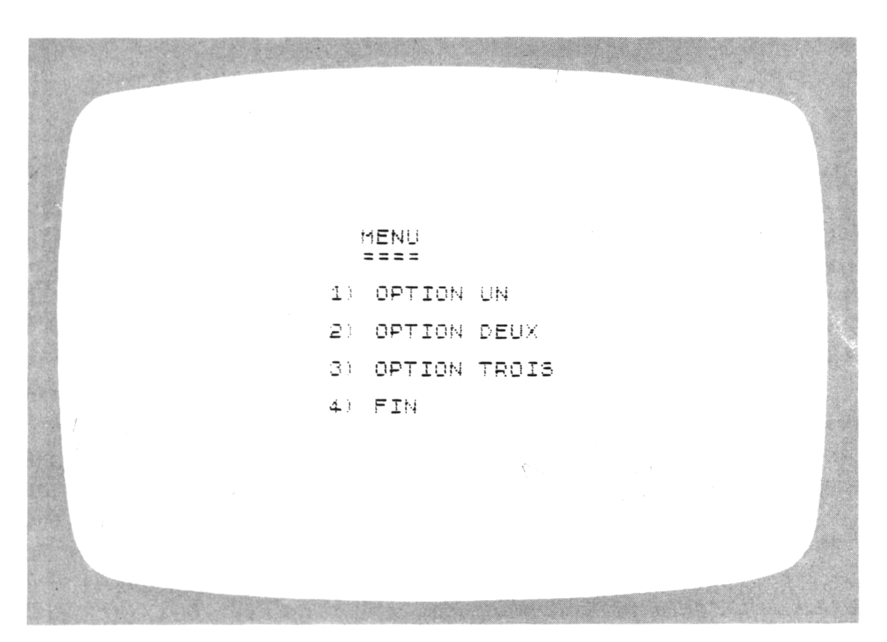


Figure G.3 : GOSUB-Exemple de résultat, "Menu"

1) JEU FACILE
2) JEU ASSEZ DIFFICILE
3) JEU TRES DIFFICILE
4) FIN

Ou bien le programme pourrait préparer un rapport financier, et proposer un menu offrant un choix de méthodes concernant un point particulier de l'élaboration du rapport :

1) AMORTISSEMENT LINEAIRE
2) AMORTISSEMENT DEGRESSIF
3) FONDS D'AMORTISSEMENT
4) FIN

Ou encore, le programme pourrait avoir des capacités de tracé de courbe et offrir à l'utilisateur le choix d'une variable et le tracé d'une fonction :

1) COURBE Y=F(X)
2) COURBE X=F(Z)
3) COURBE Z=F(Y)
4) FIN

Quelles que soient les options proposées par le programme, le rôle du menu est de présenter ces options de façon claire et d'obtenir de l'utilisateur une réponse sans équivoque exprimant son choix parmi les options. La ligne 70 lit un caractère en provenance du clavier, et la ligne 80 vérifie si ce caractère fait bien partie des choix possibles dans le menu.

```
70 INPUT M$
80 IF M$<"1" OR M$>"4"
THEN GOTO 70
```

La réponse est lue comme une chaîne de caractères, M$, (plutôt que comme une valeur numérique) de façon à éviter l'entrée de données erronées. En effet, l'ordinateur ignore toute réponse différente de l'un des quatre chiffres appropriés.

Dès que l'ordinateur a identifié un choix de menu correct, il continue en une série d'appels de sous-programmes, lignes 100 à 120. Le

second écran du programme, Figure G.2, montre les sous-programmes qu'un programme peut appeler.

Le premier appel de sous-programme est celui de la ligne 700 :

```
100 GOSUB 700
```

Ce sous-programme imprime simplement sur l'écran une rangée d'astérisques ("étoiles"). Bien que très simple, il est représentatif de nombreux sous-programmes importants que l'on peut écrire dans le seul but d'améliorer un détail d'affichage. Si l'on étudie le programme avec attention, on voit que ce sous-programme est appelé à trois reprises.

L'instruction GOSUB suivante, ligne 110, appelle un des quatre sous-programmes en option. Cette instruction est ce que l'on appelle un GOSUB *calculé* :

```
110 GOSUB (VAL M$+1)*100
```

Le sous-programme appelé par cette instruction GOSUB dépend de la valeur choisie dans le menu, M$. On verra que l'opération arithmétique spécifique qui détermine l'appel du sous-programme doit être préparée avec soin pour permettre d'arriver à l'une des quatre lignes de sous-programme : 200, 300, 400 ou 500. Voici comment l'opération se déroule :

1. La fonction VAL donne la valeur numérique de M$, choisie dans le menu (valeur comprise entre 1 et 4).

2. En ajoutant 1 à cette valeur, on obtient un nombre compris entre 2 et 5.

3. En multipliant ce nombre par 100, on obtient une des quatre adresses désirées : 200, 300, 400 ou 500.

On constate que la coordination entre les éléments essentiels de ce programme — le menu, l'instruction GOSUB calculée et les sous-programmes optionnels — n'est pas fortuite et a dû être préparée à l'avance. Le résultat de cette planification est un passage efficace du menu à l'appel de sous-programme.

On remarque que les sous-programmes optionnels ne sont que des "talons" de sous-programmes. Leur seul rôle est d'afficher un mes-

sage d'identification sur l'écran. En fait, on écrit souvent ce genre de "talon" pour garder de la place pour de futurs sous-programmes dans le cadre d'un projet d'ensemble. On peut ensuite reprendre les sous-programmes et les perfectionner.

La dernière instruction GOSUB appelle un sous-programme très important :

```
GOSUB 600
```

Ce sous-programme permet à l'utilisateur de consulter un écran d'informations aussi longtemps qu'il le désire avant de passer à l'étape suivante. Il affiche dans le coin inférieur gauche de l'écran la question :

```
CONTINUER?
```

puis attend que l'utilisateur tape sa réponse sur le clavier :

```
620 PRINT AT 21,0;"CONTINUER?"
630 INPUT A$
```

La Figure G.4 montre le résultat de ce sous-programme. Si les informations du sous-programme en option remplissent l'écran, l'utilisateur pourra l'examiner à loisir puis reprendre le déroulement du programme en appuyant sur la touche ENTER.

Notes et commentaires

- Il est toujours bon d'identifier les sous-programmes avec les lignes REM, comme le montre la Figure G.2. Il ne faut pas oublier de donner un titre approprié à chaque sous-programme.

- Certaines versions du BASIC (mais *pas* celle du ZX 81) possèdent une instruction ON...GOSUB qui remplace l'instruction GOSUB calculée. Dans ce cas, l'instruction

  ```
  110 ON VAL M$ GOSUB 200,300,400,500
  ```

  aura le même effet que l'instruction GOSUB calculée dans la ligne 110 du programme.

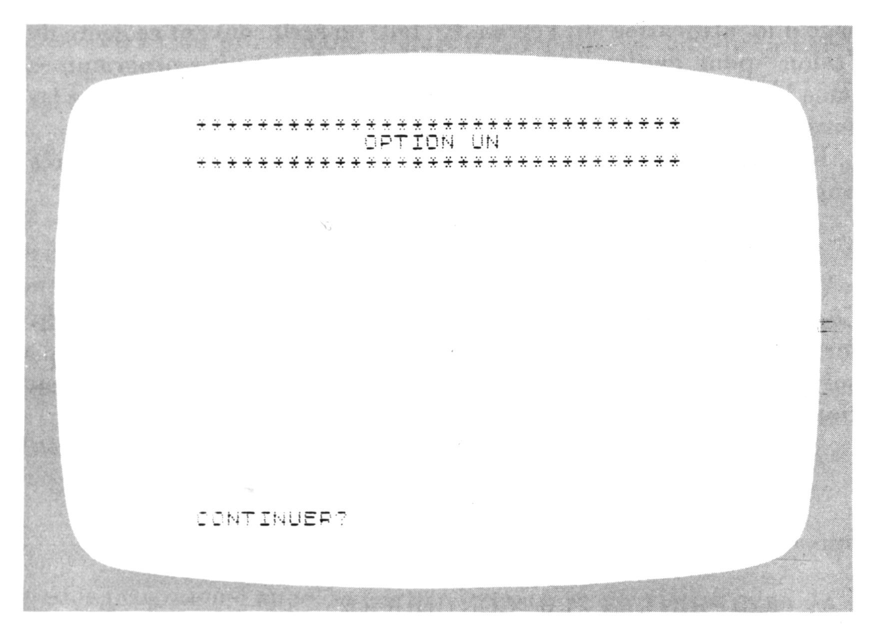


Figure G.4 : GOSUB-Exemple de résultat (suite)

GOTO (mot clé ; touche [G])

L'instruction GOTO dévie le déroulement du programme en cours vers une ligne définie. Le numéro de la ligne comme dans l'instruction GOSUB peut être exprimé comme une valeur littérale :

GOTO 10

une variable :

GOTO L

ou une expression arithmétique :

GOTO (L+5)*100

L'instruction GOTO peut être utilisée en commande immédiate ou

comme une instruction dans une ligne de programme. (Pour une utilisation importante de GOTO en commande immédiate, voir la rubrique SAVE.) Si l'instruction GOTO est insérée dans le programme, le "saut" qu'elle engendre peut être un retour en arrière :

```
90 GOTO 10
```

ou un "saut" en avant :

```
100 GOTO 150
```

ou même un retour au début de la ligne sur laquelle elle se trouve :

```
20 IF INKEY$= "" THEN GOTO 20
```

La ligne 20, ci-dessus, qui sera répétée jusqu'à la frappe de n'importe quelle touche du clavier, illustre l'utilisation de GOTO dans une instruction IF. Dans une instruction GOTO *soumise à condition*, le saut ne sera exécuté que si l'expression logique contenue dans l'instruction IF est "vraie".

Exemple de programme

Le programme de la Figure G.5 aide l'utilisateur à gérer son compte bancaire en lui donnant un enregistrement des retraits et l'état de son compte après chaque opération. Le programme commence par lui demander son solde antérieur et le numéro du dernier chèque qui a été enregistré. Ces deux valeurs sont attribuées respectivement aux variables B et N.

Les instructions des lignes 80 à 120 lui demandent quel est le montant d'un chèque donné et affichent une ligne d'informations donnant le numéro du chèque, son montant et l'état du compte. On stocke dans la variable A les montants successifs des chèques. La variable I compte le nombre de chèques traités par le programme.

Le programme contient deux instructions GOTO importantes. L'instruction de la ligne 130 renvoie le contrôle du programme au début du bloc d'instructions qui traite chaque chèque :

```
130 GOTO 80
```

```
  10 PRINT AT 21,0;"SOLDE ANTERI
EUR ?"
  20 INPUT B
  30 PRINT AT 21,0;"NO. DU DERNI
ER CHEQUE ?"
  40 INPUT N
  50 CLS
  55 LET I=1
  60 PRINT "NUMERO     MONTANT
ETAT"
  80 PRINT AT 20,0;"CHEQUE NO. "
;N+I
  90 PRINT AT 21,0;"MONTANT ?"
 100 INPUT A
 110 IF A>B THEN GOTO 140
 115 LET B=B-A
 120 PRINT AT I+1,0;N+I;AT I+1,1
2;A;AT I+1,22;B
 125 LET I=I+1
 130 GOTO 80
 140 PRINT AT I+1,0;N+I;AT I+1,1
2;A;AT I+1,20;"* CHEQUE "
 150 PRINT AT I+2,22;"EN BOIS *"
```

Figure G.5 : GOTO-Exemple de programme

Cette instruction forme ainsi une boucle qui permet de traiter un nombre illimité de chèques.

Cependant, l'instruction GOTO de la ligne 110 pare à l'éventualité d'un découvert. Cette instruction GOTO fait partie d'une instruction IF :

110 **IF** A>B **THEN GOTO** 140

Cette ligne signifie : "Si le montant du chèque en cours de traitement est supérieur au montant du solde disponible, alors aller à la ligne 140". Ce saut est *extérieur* à la boucle de traitement de chèques formée par l'instruction GOTO de la ligne 130. La ligne 140 affiche sur l'écran un "message de découvert", puis le programme s'arrête.

La Figure C.6 simule le déroulement de ce programme.

Notes et commentaires

- On peut utiliser l'instruction GOTO en commande immédiate

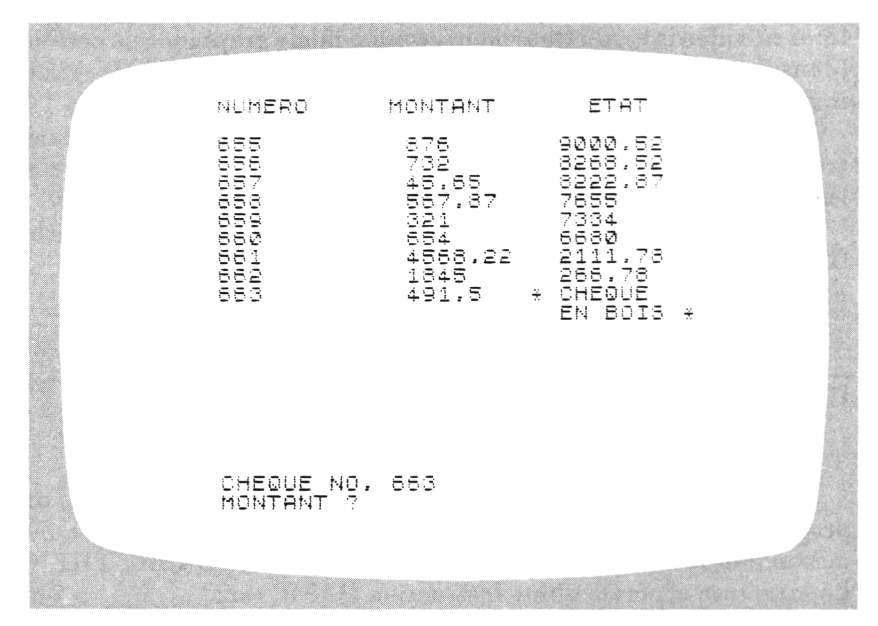


Figure G.6 : GOTO-Exemple de déroulement de programme

pour indiquer à l'ordinateur le point de départ du programme. Par exemple, si le programme chargé dans la mémoire de l'ordinateur commence à la ligne 10, on peut entrer la commande

GOTO 10

pour démarrer le programme. Il est préférable d'utiliser GOTO de cette façon pour ne perdre aucune des variables obtenues lors d'un déroulement antérieur du programme. (Il ne faut pas oublier que la commande RUN, alternative du cas ci-dessus, exécute un CLEAR — initialisation — de toutes les variables avant de commencer le programme).

GRAPHICS (shift ; touche [9])

La touche GRAPHICS permet de placer le clavier en mode graphique, c'est-à-dire de taper des caractères graphiques ou des carac-

tères en vidéo inverse. Quand on passe en mode graphique, le curseur (dans la zone de travail de l'écran) devient un "G" en vidéo inverse.

Les caractères graphiques situés sur les première, deuxième et troisième rangées du clavier sont obtenus avec la touche SHIFT. Les lettres ou chiffres en vidéo inverse sont obtenus normalement.

Pour sortir du mode graphique, il faut taper à nouveau la touche GRAPHICS (SHIFT-9).

IF (mot clé ; touche [U])

IF est un mot clé important qui permet de donner à un programme BASIC la possibilité de faire des choix. La syntaxe d'une instruction IF nécessite l'utilisation d'un autre mot BASIC, THEN. La structure générale d'une instruction BASIC est :

IF (expression logique) **THEN** (exécution du mot clé)

Quand l'ordinateur exécute une instruction IF, il évalue l'expression logique comme étant vraie ou fausse. Si l'expression est vraie, l'ordinateur exécute l'instruction qui suit le mot THEN. Si l'expression est fausse, l'instruction IF n'a aucune conséquence ; l'ordinateur passe à l'instruction suivante.

Les expressions logiques sont des relations d'égalité ou d'inégalité qui sont soit vraies soit fausses. On peut écrire ces expressions en utilisant un ou plusieurs des symboles suivants :

=	*est égal à*
< >	*est différent de*
<	*est inférieur à*
>	*est supérieur à*
< =	*est inférieur ou égal à*
> =	*est supérieur ou égal à*

Les mots BASIC AND, OR et NOT peuvent aussi être utilisés pour construire ou modifier des expressions logiques. (Pour plus d'information, voir ces rubriques.)

Dans une instruction IF, on peut mettre n'importe quelle commande du BASIC ZX 81 après le mot THEN.

Voici trois exemples d'instructions IF suivies de leurs transcriptions en langage courant :

```
IF HEURE>12 THEN LET HEURE=HEURE-12
```

"Si la variable HEURE contient une valeur supérieure à 12, alors stocker une nouvelle valeur dans HEURE, égale à HEURE moins 12".

```
IF AGE=65 THEN GOSUB 300
```

"Si la variable AGE contient la valeur 65, alors exécuter le sous-programme à la ligne 300".

```
IF T< =N THEN INPUT N
```

"Si la valeur de T est inférieure ou égale à la valeur de N, alors lire une nouvelle valeur de N en provenance du clavier".

Exemple de programme

Le programme de la Figure I.1 est une version du jeu de devinettes classique Devin adaptée au ZX 81. Dans cette version du jeu, l'ordinateur choisit au hasard un nombre compris entre 1 et 100 et donne au joueur sept chances de le deviner. Après chaque proposition du joueur, l'ordinateur lui indique si le nombre proposé est supérieur ou inférieur au nombre à trouver.

Il y a au cœur de ce programme une série d'instructions IF qui permettent à l'ordinateur d'évaluer la proposition et de choisir sa réponse.

Les lignes 10 à 55 affichent un jeu d'instructions en haut de l'écran. La ligne 60 utilise la fonction RND dans une formule qui donne à l'ordinateur un nombre compris entre 1 et 100 inclus. Ce nombre est assigné à la variable N. La ligne 70 crée un compteur (I) qui comptabilise le nombre d'essais du joueur. Enfin, la ligne 90 lit la proposition en provenance du clavier et l'attribue à la variable G pour que le programme puisse la comparer au nombre à trouver.

```
  10 PRINT "* DEVIN *"
  20 PRINT
  30 PRINT "JE PENSE A UN NOMBRE
"
  40 PRINT "COMPRIS ENTRE 1 ET 1
00, VOUS AVEZ"
  50 PRINT "7 ESSAIS POUR LE DEV
INER"
  55 PRINT
  60 LET N=1+INT (RND*100)
  70 LET I=1
  80 PRINT I;": ";
  90 INPUT G
 100 PRINT G;" ==> ";
 110 IF G=N THEN GOTO 180
 120 IF G>N THEN PRINT "SUPERIEU
R"
 130 IF G<N THEN PRINT "INFERIEU
R"
 140 LET I=I+1
 150 IF I<=7 THEN GOTO 80
 160 PRINT AT 15,0;"DESOLE, LE N
OMBRE ETAIT ";N;"."
 170 STOP
 180 PRINT "** BRAVO **"
```

Figure I.1 : IF-Exemple de programme

Le premier test vérifie si le joueur a deviné le nombre :

> 110 **IF** G = N **THEN GOTO** 180

Si le joueur a deviné, le programme passe à la dernière ligne, ligne 180, qui lui annonce qu'il a trouvé la bonne réponse.

Si la réponse est incorrecte, les deux instructions IF qui suivent affichent sur l'écran soit "SUPERIEUR", soit "INFERIEUR" :

> 120 **IF** G > N **THEN PRINT** "SUPERIEUR"
> 130 **IF** G < N **THEN PRINT** "INFERIEUR"

Ensuite le compteur I est incrémenté de 1 et une instruction IF vérifie si le joueur a épuisé toutes ses chances.

> 150 **IF** I < = 7 **THEN GOTO** 80

S'il a encore la possibilité de jouer, alors le programme retourne à la

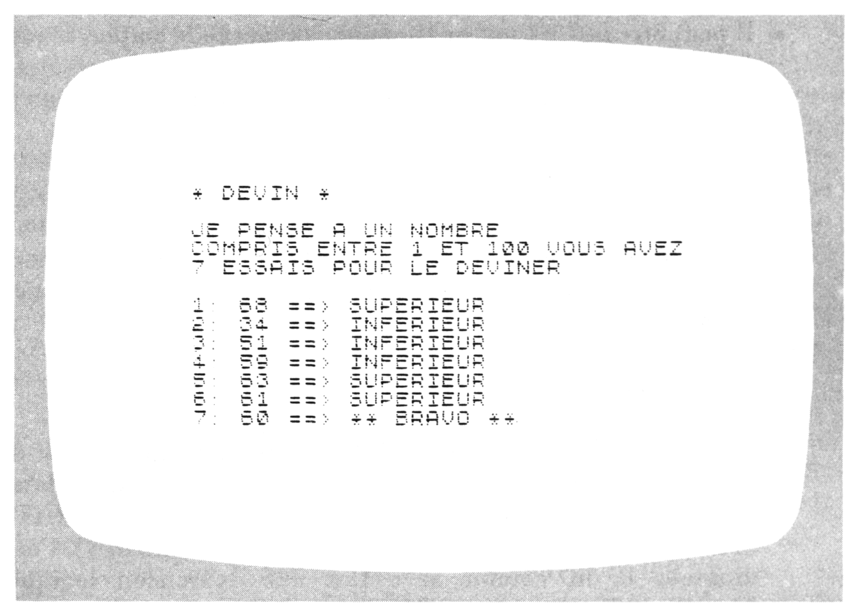


Figure I.2 : IF-Exemple de résultat

ligne 80 pour recommencer le processus. Sinon, si l'expression :

I< =7

est fausse, alors la ligne 150 est ignorée et le programme termine avec le message de condoléances de la ligne 160.

Notes et commentaires

- Certaines versions du BASIC ont une instruction supplémentaire, ELSE, utilisable avec l'instruction IF, ce qui permettrait dans l'exemple précédent de combiner les lignes 120 et 130 :

```
120 IF G > N THEN PRINT "SUPERIEUR"
ELSE PRINT "INFERIEUR"
```

- Il peut être intéressant de connaître la méthode employée par le ZX 81 pour évaluer des expressions logiques. Le résultat d'une telle évaluation est codé numériquement comme suit :

  vrai = 1
  faux = 0

  Cela signifie qu'il est possible dans une instruction IF de remplacer l'expression logique par une simple variable numérique :

  IF N **THEN PRINT** "BONJOUR"

  Si la valeur de N est égale à 0, l'ordinateur réagit comme si une expression logique "fausse" avait été introduite dans l'instruction IF. Dans ce cas, il ignore l'instruction PRINT. Si N a une valeur différente de 0 (mais pas seulement 1), l'ordinateur la lit comme si c'était une expression logique "vraie", et l'instruction PRINT (située à la fin de l'instruction IF) est exécutée.

INKEY$ (fonction ; touche [B])

A chaque exécution de la fonction INKEY$, l'ordinateur scrute le clavier pour voir si une touche a été tapée. Dans l'affirmative, la fonction INKEY$ donne le caractère en mode lettre de la touche pressée. Sinon, il donne une chaîne de caractères vide ("nul").

Etant donné que l'ordinateur scrute le clavier très rapidement, on n'a pas le temps, si l'instruction INKEY$ n'est exécutée qu'une fois, d'appuyer sur une touche. C'est la raison pour laquelle on intègre généralement la fonction à une boucle pour que l'ordinateur scrute le clavier plusieurs fois. Voici un exemple :

```
10 LET L$ = INKEY$
20 IF L$ = "" THEN GOTO 10
30 PRINT "VOUS AVEZ APPUYE SUR LA TOUCHE" ;L$
40 GOTO 10
```

Ce petit programme affichera le message suivant

VOUS AVEZ APPUYE SUR LA TOUCHE T

à chaque fois que l'on appuiera sur la touche T du clavier. On remarque que la fonction INKEY$ fait partie de l'instruction LET de la ligne 10. Cette instruction assigne à la chaîne de caractères L$ n'importe quel caractère donné par la fonction INKEY$. La ligne 20 teste la valeur de L$. Si cette valeur est une chaîne vide (""), l'instruction GOTO renvoie le programme à la ligne 10 pour une nouvelle exécution de la fonction INKEY$. L'ordinateur scrute donc le clavier jusqu'à ce qu'une touche soit tapée.

La fonction INKEY$ *n'envoie pas* systématiquement un "écho" (c'est-à-dire un affichage sur l'écran) de la valeur de la touche frappée. C'est une des différences importantes avec la fonction INPUT. La fonction INPUT envoie un écho de chaque touche tapée sur le clavier en affichant le caractère qui lui correspond dans l'espace de travail situé dans le bas de l'écran. La fonction INKEY$ laisse au programmeur la décision de l'affichage d'un caractère et le choix de son emplacement.

Exemple de programme

Un bon programmeur tient compte, lors de l'entrée de données, des possibilités d'affichage de l'écran. Les lignes de la Figure I.3 (qui ne sont qu'un fragment de programme) ont pour fonction de réaliser un affichage propre et clair en réponse aux données introduites au clavier. En étudiant l'algorithme représenté par ces lignes, on s'aperçoit que la fonction INKEY$, quoique beaucoup plus difficile à employer que l'instruction INPUT, donne par contre, au cours du programme, des possibilités d'affichages mieux présentés.

On peut voir les possibilités offertes par ces lignes en faisant tourner le programme. Il commence par afficher sur l'écran une rangée de lettres et une rangée de chiffres. Il est prévu pour lire alternativement une lettre et un chiffre. Le titre quelque peu curieux "SELECTION DE DISQUE" est affiché en haut de l'écran comme si la combinaison lettre-chiffre opérait une sélection de "juke-box". Cependant, ce type d'affichage est intéressant dans de nombreux cas de programmation lorsque l'utilisateur doit faire un choix de menu

```
  10 PRINT TAB 5;"SELECTION DE D
ISQUES"
  20 FOR I=1 TO 10
  30 PRINT AT 5,4+2*I;CHR$ (I+37
)
  40 PRINT AT 8,4+2*I;CHR$ (I+27
)
  50 NEXT I
  60 LET L$=INKEY$
  70 IF NOT (L$>="A" AND L$<="J"
) THEN GOTO 60
  80 PRINT AT 5,4+2*(CODE L$-37)
;CHR$ (CODE L$+128)
  90 LET D$=INKEY$
 100 IF NOT (D$>="0" AND D$<="9"
) THEN GOTO 90
 110 PRINT AT 8,4+2*(VAL D$+1);C
HR$ (CODE D$+128)
 120 PRINT AT 21,0;"CONTINUER ?"
 130 INPUT A$
 140 CLS
 150 GOTO 10
```

Figure I.3 : INKEY$-Exemple de programme

qui le dirigera vers une certaine étape du programme (voir l'exemple de programme de la rubrique GOSUB).

La Figure I.4 montre l'affichage initial alors que le programme attend l'entrée de données. La première donnée lue par le programme est une lettre comprise entre A et J. Une touche correspondant à un caractère en dehors de ces limites sera ignorée. Par contre, si l'utilisateur appuie sur l'une de ces touches, la lettre correspondante sera transformée sur l'écran en caractère vidéo inverse. L'ordinateur attend ensuite un chiffre compris entre 0 et 9. Comme pour les lettres, si la touche tapée se trouve dans ces limites, le chiffre correspondant se transforme en caractère vidéo inverse. La Figure I.5 montre un affichage après la sélection "B7".

Après la sélection de la combinaison lettre-chiffre, le programme est vraisemblablement prêt à se diriger dans la direction choisie. Cet exemple de programme pose la question :

CONTINUER ?

dans le coin inférieur gauche de l'écran et attend que l'utilisateur

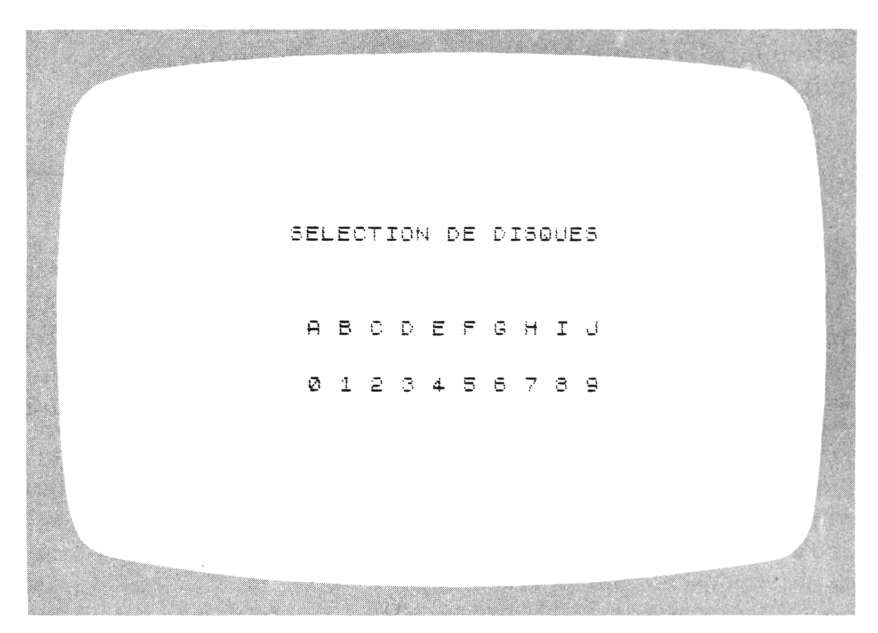


Figure I.4 : INKEY$-Exemple de résultat, avant réponse au clavier

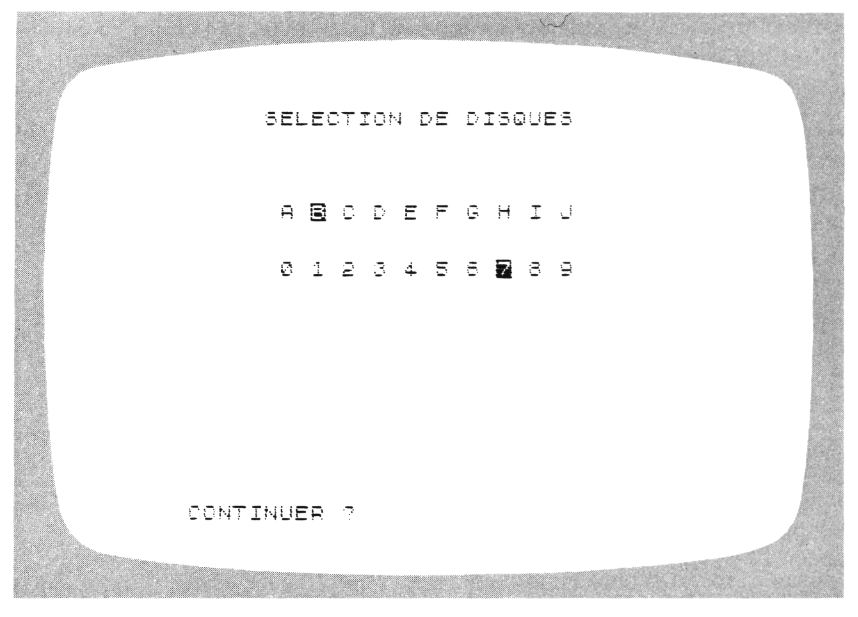


Figure I.5 : INKEY$-Exemple de résultat, après réponse au clavier

tape la touche ENTER pour reprendre l'ensemble du processus de sélection. Si cet exemple faisait partie d'un programme réel, on pourrait trouver à la place une autre question :

```
ACCEPTEZ-VOUS CETTE SELECTION ?
(O) OU (N)
```

qui permettrait à l'utilisateur de confirmer cette sélection ou bien de l'annuler et de recommencer le processus.

Si l'on regarde le programme sur la Figure I.3, et plus particulièrement la fonction INKEY$ (lignes 60 et 90), on voit que la ligne 60 assigne le résultat de la fonction INKEY$ à la variable L$. Si L$ est une chaîne vide ou bien un caractère situé en dehors des limites "A" à "J", la ligne 70 renvoie à la ligne 60 :

```
70 IF NOT (L$> ="A" AND L$< ="J")
THEN GOTO 60
```

De la même manière, la ligne 100 renvoie à la ligne 90 jusqu'à ce que la fonction INKEY$ lise un chiffre compris entre 0 et 9. Le programme est capable, grâce à ces deux boucles, de sélectionner des combinaisons appropriées.

Les lignes 80 et 110 ont pour tâche de disposer, à leur place, les caractères en vidéo inverse sur l'écran. Ce travail nécessite l'utilisation des fonctions CHR$ et CODE. Pour plus de détails, on étudiera soigneusement le code caractère du BASIC ZX 81 (voir les rubriques CHR$ et CODE).

INPUT (mot clé ; touche [I])

La commande INPUT donne l'ordre à l'ordinateur d'attendre une donnée en provenance du clavier. Une fois la donnée introduite (c'est-à-dire quand l'utilisateur a appuyé sur la touche ENTER),

l'ordinateur assigne cette donnée à la variable désignée dans l'instruction INPUT. Cette instruction prend la forme :

INPUT V

dans laquelle V est le nom d'une variable numérique, ou bien :

INPUT V$

dans laquelle V$ est le nom d'une chaîne de caractères. On peut utiliser l'instruction INPUT pour initialiser une *variable* si celle-ci n'a pas été utilisée antérieurement dans le programme.

L'instruction INPUT entraîne un affichage particulier qui dépend du type de variable défini dans l'instruction (variable numérique ou chaîne de caractères). On voit sur la Figure I.6 le signal réclamant l'entrée d'une chaîne de caractères : le curseur-L, entre guillemets, apparaît dans le coin en bas à gauche de l'écran. A chaque fois que l'on tape un caractère sur le clavier, celui-ci apparaît sur l'écran entre les premiers guillemets et le curseur-L (voir Figure I.7). Pour finir, le signal et les données disparaissent quand on appuie sur la touche ENTER.

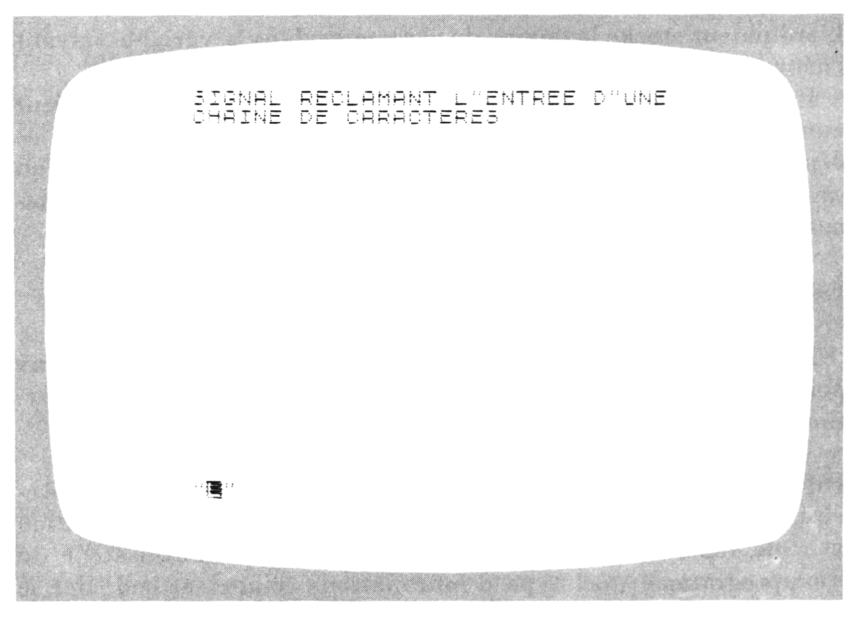


Figure I.6 : Repère de chaîne de caractères

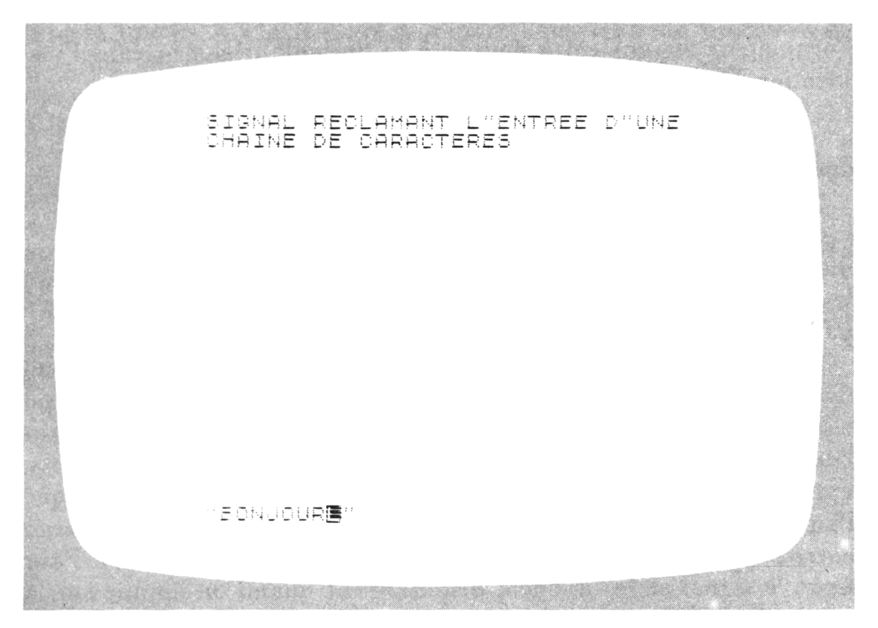


Figure I.7 : Entrée d'une chaîne de caractères

L'ordinateur stocke la chaîne de caractères dans la variable suivant l'instruction INPUT.

La Figure I.8 montre le signal réclamant l'entrée d'une valeur numérique : curseur-L sans guillemets. Dans ce cas, le curseur se déplace d'un espace vers la droite à chaque fois que l'on tape un caractère sur le clavier et celui-ci apparaît sur l'écran immédiatement à gauche du curseur.

Exemple de programme

Le programme de la Figure I.9 est un exercice qui illustre deux points différents mais importants, relatifs à l'utilisation de l'instruction INPUT :

1. Quand on écrit un programme qui lit des données en provenance du clavier, on provoque à certains moments l'apparition de repères indicateurs ; il faut donc dire à l'utilisateur du programme quel type d'informations celui-ci attend. Sur le ZX 81, la meilleure place pour ce type de message est, en géné-

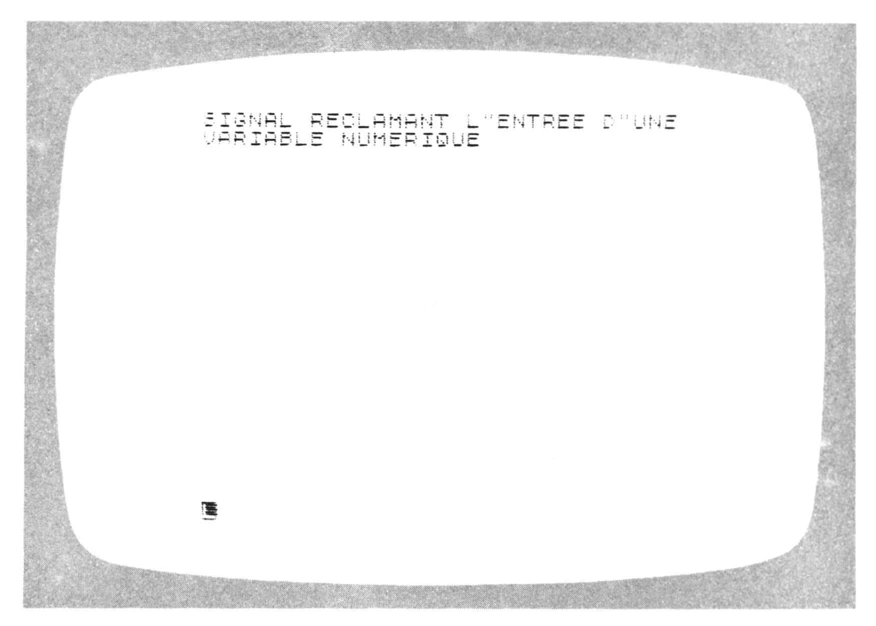


Figure I.8 : INPUT-Repère d'entrée numérique

ral, en bas de l'écran près de l'espace de travail. En d'autres termes, le repère indicateur doit apparaître aussi près que possible de la ligne d'édition, là où sont affichées les données en provenance du clavier.

2. L'instruction INPUT permet d'entrer des expressions arithmétiques aussi bien que des données numériques. Dans ce cas, l'ordinateur calcule l'expression puis attribue le *résultat* à la variable numérique définie dans l'instruction INPUT. Ces expressions arithmétiques peuvent contenir des variables si celles-ci ont été définies et initialisées auparavant dans le programme.

Les lignes 10 à 60 (Figure I.9) attribuent au hasard des valeurs aux variables numériques V1, V2 et V3, puis les affichent en haut de l'écran. Les lignes 70 et 80 (juste avant l'instruction INPUT de la ligne 90) ont pour tâche de fournir des repères indicateurs. Ces lignes utilisent toutes les deux la fonction AT pour afficher les repères en bas de l'écran (aux adresses 20,0 et 21,0).

```
  10 LET V1=RND
  20 LET V2=RND
  30 LET V3=RND
  40 PRINT "V1 = ";V1
  50 PRINT "V2 = ";V2
  60 PRINT "V3 = ";V3
  70 PRINT AT 17,0;"TAPER UNE EX
PRESSION "
  75 PRINT AT 18,0;"ARITHMETIQUE
 QUELCONQUE"
  80 PRINT AT 19,0;"UTILISANT CE
S TROIS VARIABLES:"
  90 INPUT V4
 100 PRINT AT 10,0;"
                    "
 110 PRINT AT 10,0;"RESULTAT =
";V4
 120 GOTO 90
```

Figure I.9 : INPUT-Exemple de programme

La ligne 90 lit une valeur d'entrée dans la variable V4, et la ligne 110 l'affiche comme "RESULTAT". Ceci est illustré par deux écrans de résultats (Figures I.10 et I.11). Dans la Figure I.10, on a tapé l'expression arithmétique suivante :

100*(V1 + V2)* ASN V3

Le repère, toujours placé à la fin de l'expression, indique qu'elle n'a pas encore été validée. On remarque que l'expression est composée d'une valeur littérale (100), d'une fonction (ASN), de deux opérations *, +) et des trois variables définies (V1, V2 et V3). *Toute*, expression arithmétique légale est acceptée comme une entrée numérique.

La Figure I.11 montre l'écran après validation de la ligne (touche ENTER). Comme prévu, l'ordinateur a calculé l'expression arithmétique, attribué le résultat de l'opération à la variable V4 puis l'a affiché sur l'écran.

Puis, l'ordinateur invite l'utilisateur à entrer une nouvelle expression arithmétique et le même processus se déroule (voir ligne 90). Puisque la variable V4 a déjà été définie, elle peut être utilisée dans la nouvelle expression arithmétique.

```
V1 = 0,99543762
V2 = 0,65782166
V3 = 0,33700562




TAPER UNE EXPRESSION
ARITHMETIQUE QUELCONQUE
UTILISANT CES TROIS VARIABLES:

100*(V1+V2)*ASN V3
```

Figure I.10 : INPUT-Exemple de résultat, 1

```
V1 = 0,38174439
V2 = 0,63153076
V3 = 0,36521912


RESULTAT = 50,364351


TAPER UNE EXPRESSION
ARITHMETIQUE QUELCONQUE
UTILISANT CES TROIS VARIABLES:
```

Figure I.11 : INPUT-Exemple de résultat, 2

Notes et commentaires

- L'instruction INPUT ne peut pas être utilisée en commande immédiate. L'ordinateur donne alors un message d'erreur "8".

- Si l'ordinateur attend une entrée numérique et que l'on introduit une séquence alphanumérique commençant par une lettre qui ne correspond pas à un nom de variable défini dans le programme en cours, celui-ci s'arrête avec un message d'erreur "2". Cela signifie que l'on a essayé d'utiliser une variable non définie.

INT (fonction ; touche [R])

La fonction INT donne la valeur entière d'un nombre. Si le nombre est positif, la fonction INT se contente d'éliminer sa partie fractionnaire. Par exemple, l'expression :

```
INT 2.7
```

a pour résultat la valeur 2.

Si c'est un nombre négatif, l'instruction INT du ZX 81 donne la valeur entière inférieure la plus proche. Par exemple :

```
INT -2.7
```

a pour résultat la valeur -3.

Exemple de programme

Le programme de la Figure I.12 compare pour une série d'arguments donnée le résultat de la fonction INT et celui obtenu par une *formule* d'arrondi, la formule suivante est utilisée pour arrondir un nombre, N, au nombre entier le plus proche :

```
  10 LET T1=15
  20 LET T2=25
  30 PRINT "NOMBRE";TAB T1;"INT"
;TAB T2-3;"ARRONDI"
  40 PRINT
  50 FOR N=-1.8 TO 1.8 STEP 0.2
  60 PRINT N;TAB T1;INT N;TAB T2
;INT (N+.5)
  70 NEXT N
```

Figure I.12 : INT-Exemple de programme

INT (N+.5)

La variable N est la variable de contrôle d'une boucle FOR, elle varie entre -1,8 et +1,8 par pas (step) de 0,2 comme le montre le résultat du programme (Figure I.13).

Notes et commentaires

- On remarque quelque chose d'inhabituel dans le résultat de la Figure I.13. Quand N atteint ce qui devrait être la valeur 0, une "erreur cumulée"apparaît sur l'écran. On voit à la place de 0 un nombre très voisin de 0 exprimé en notation exponentielle :

  6.9849193E-10

Cette erreur est due à la manière dont le ZX 81 stocke et traite les nombres dans le système appelé *notation en virgule flottante (floating-point binary)*. "L'erreur cumulée"a une

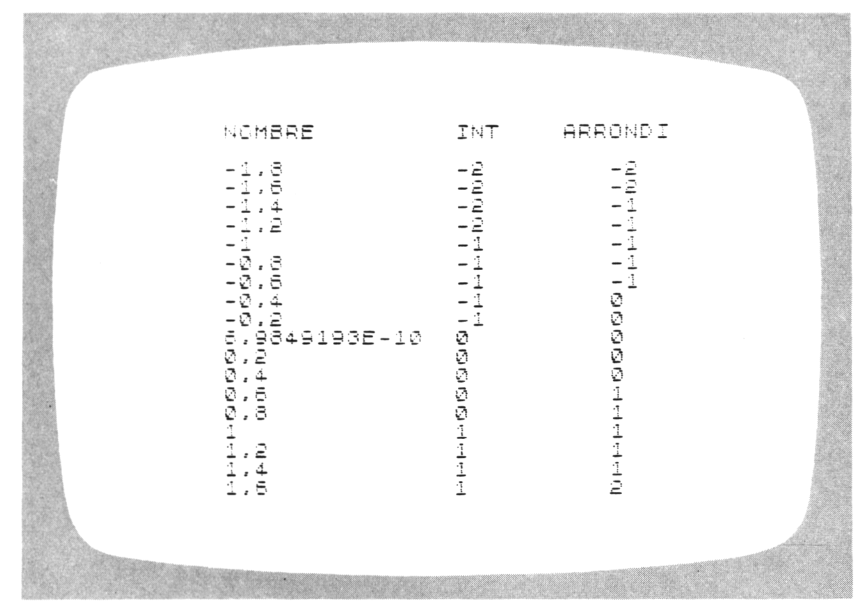


Figure I.13 : INT-Exemple de résultat

autre conséquence : la dernière valeur de N (1,8) n'est pas prise en compte par la boucle FOR. Il faut être vigilant car ce type d'erreurs peut apparaître lors d'applications numériques.

LEN (fonction ; touche [K])

La fonction LEN (*length*, longueur) nécessite un argument de chaîne et donne un nombre entier qui représente la longueur de la chaîne, en caractères. Par exemple, l'expression :

LEN "BONJOUR"

a pour résultat la valeur 7 parce que "BONJOUR" est constitué de 7 caractères.

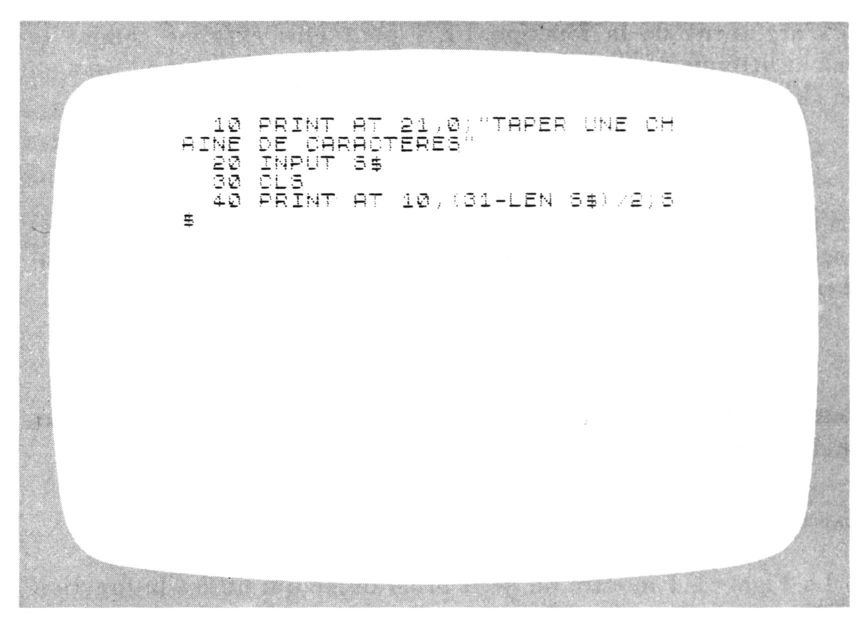


Figure L.1 : LEN-Exemple de programme

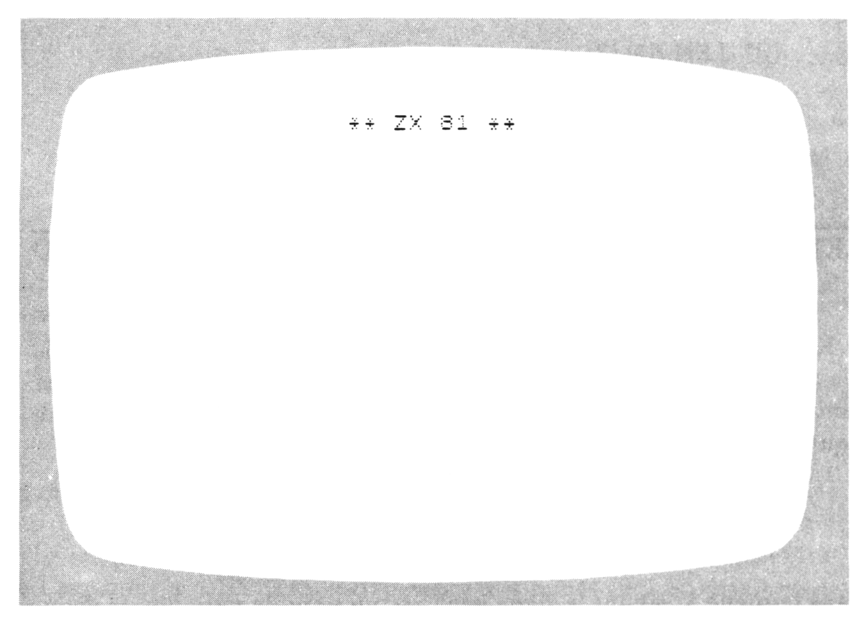


Figure L.2 : LEN-Exemple de résultat

L'argument de la fonction LEN peut être exprimé comme la valeur littérale de chaîne (voir exemple ci-dessus), ou comme un nom de chaîne de caractères :

LEN S$

La fonction LEN accepte également comme argument une concaténation de deux chaînes ou plus qui doit être placée entre parenthèses :

LEN (S$ + G$)

Cette expression a pour résultat la somme des longueurs des deux chaînes S$ et G$.

Exemple de programme

La Figure L.1 montre un petit programme qui utilise la fonction LEN pour centrer une chaîne sur l'écran. La formule de mise en forme fait partie d'une adresse AT (ligne 40) :

(31 - **LEN** S$)/2

La Figure L.2 montre le résultat de ce programme.

LET (mot clé ; touche [L])

L'instruction LET assigne une valeur à une variable. Si la variable n'existe pas encore dans le programme, la fonction LET la crée et l'initialise. Si la variable existe déjà, la fonction LET lui attribue une nouvelle valeur.

L'instruction LET prend la forme suivante :

LET V = valeur

dans laquelle V, avant le signe égal, est un nom de variable quelconque (variable numérique ou chaîne de caractères). La valeur située

après le signe égal peut être exprimée par une valeur littérale, une variable ou une expression composée de valeurs littérales et/ou de variables. L'instruction LET donne l'ordre à l'ordinateur d'évaluer l'expression qui se trouve après le signe égal et de stocker le résultat dans la zone mémoire correspondant à la variable située avant le signe égal. L'instruction LET est parfois appelée *instruction d'affectation*. Voici trois exemples commentés :

```
LET AGE=18
```

"stocker la valeur 18 dans la variable AGE."

```
LET I=J
```

"stocker la valeur de la variable J dans la variable I." (Il faut avoir assigné une valeur à J avant l'exécution de cette instruction. La valeur de J *ne change pas* au cours de cette exécution.)

```
LET N=5*M+P/2
```

"Evaluer l'expression située après le signe égal et stocker le résultat dans la variable N."(Il faut avoir assigné des valeurs à M et à P avant l'exécution de cette instruction. Les valeurs de M et de P *ne changent pas* au cours de cette exécution.)

On remarque qu'il n'y a pas de limite à la complexité de l'expression située après le signe égal. Par contre il n'y a jamais plus d'un nom de variable avant.

L'instruction LET peut être exécutée soit en commande immédiate, soit insérée dans un programme.

Exemple de programme

Le programme de la Figure L.3 montre des utilisations différentes de l'instruction LET. Les instructions des lignes 10 à 50 attribuent des chaînes de caractères aux cinq éléments du tableau de chaînes de caractères, L$. On remarque que dans chacune des instructions LET

```
  5 DIM L$(5,9)
 10 LET L$(1)="PREMIERE"
 20 LET L$(2)="SECONDE"
 30 LET L$(3)="TROISIEME"
 40 LET L$(4)="QUATRIEME"
 50 LET L$(5)="CINQUIEME"
 60 LET I=1
 70 IF I>5 THEN GOTO 110
 74 LET V=I*3
 76 LET H=I*2
 80 PRINT AT V,H;I;". ";L$(I);"
 BOUCLE"
 90 LET I=I+1
100 GOTO 70
110 STOP
```

Figure L.3 : LET-Exemple de programme

le nom de la variable située à gauche du signe égal est en fait le nom d'un élément du tableau. Alors que ces instructions semblent attribuer des chaînes de caractères de longueurs variables aux éléments de L$, l'instruction DIM (ligne 5) spécifie que chaque élément de L$ aura exactement 9 caractères. Par exemple, la ligne 10.

```
10 LET L$(1)="PREMIERE"
```

aura pour effet d'attribuer la chaîne de 9 caractères

"PREMIERE"

à l'élément L$(1). Pour plus d'information sur ce point, voir la rubrique DIM.

Les lignes 60, 70, 90 et 100 de ce programme exécutent un travail identique à celui que réalise une boucle FOR. La ligne 60 initialise la variable I à la valeur 1. Cette variable servira de *compteur* dans la boucle.

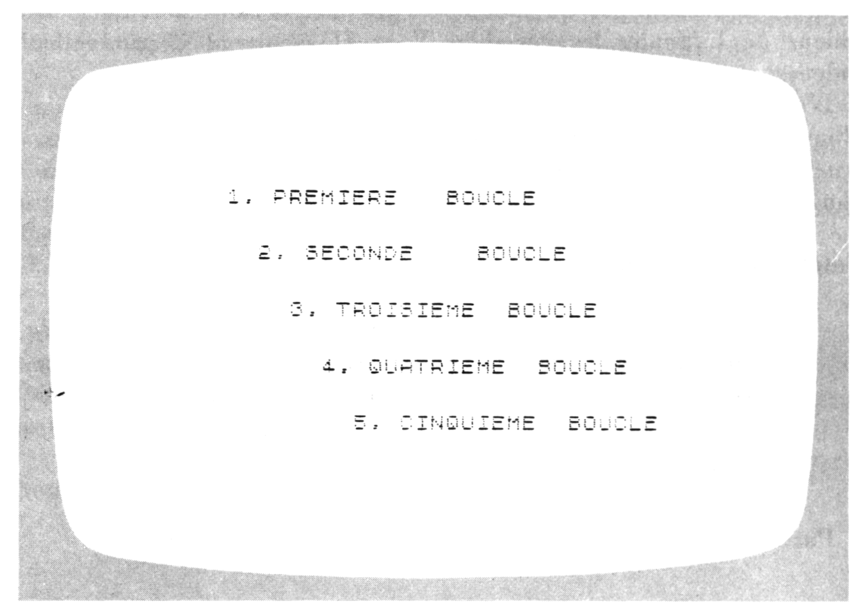


Figure L.4 : LET-Exemple de résultat

L'instruction LET (ligne 90) *incrémente* de 1 la valeur de I, à chaque itération.

```
90 LET I=I+1
```

Ce genre d'instruction semble souvent obscur au programmeur débutant ; on peut la transcrire comme suit :

"Ajouter 1 à la valeur courante de I ; puis stocker à nouveau dans I la nouvelle valeur incrémentée."

L'ancienne valeur de I a disparu.

Les instructions LET (lignes 74 et 76) déterminent les coordonnées verticales et horizontales de l'adresse AT à utiliser dans l'instruction PRINT de la ligne 80 :

```
74 LET V=1*3
76 LET H=1*2
```

Il ne faut pas oublier qu'aucune de ces instructions ne change la valeur de I. Seules les variables V et H reçoivent de nouvelles valeurs.

L'utilisateur étudiera le résultat de ce programme avec attention (Figure L.4). Il doit s'assurer d'avoir bien compris comment l'instruction LET (ligne 90) contrôle dans la boucle les paramètres d'affichage.

Notes et commentaires

- Dans certaines versions du BASIC, le mot LET est facultatif dans une instruction d'assignation. On peut donc trouver dans certains programmes en BASIC des instructions telles que :

```
90 I=I+1
```

Par contre le mot LET est obligatoire en BASIC ZX 81.

Ligne de programme en cours (vocabulaire informatique-ZX 81)

La ligne en cours est celle qui est marquée par un symbole (>) en vidéo inverse ; si l'on tape la touche EDIT (SHIFT et 1), une reproduction de la ligne en cours descend dans l'espace de travail de l'écran, prête à être modifiée. Normalement, la ligne en cours est la ligne de programme que l'on vient d'entrer, mais on peut la changer en appuyant sur les touches flèches verticales (montante : SHIFT-7, descendante : SHIFT-6). A chaque fois que l'on appuie sur l'une de ces touches, le curseur monte ou descend d'une ligne (voir la rubrique EDIT).

LIST (mot clé ; touche [K])

La commande LIST provoque l'affichage sur l'écran du programme stocké en mémoire. Bien que cette commande puisse être

insérée dans un programme, on l'utilise généralement en commande immédiate. Elle peut prendre deux formes. La première est :

LIST

Cette commande provoque l'affichage du programme, de la première à la dernière ligne ou de 22 lignes qui correspondent à la capacité maximum de l'écran. La deuxième forme est :

LIST L

dans laquelle L est un numéro de ligne du programme. Dans ce cas l'affichage commence à la ligne L et continue soit jusqu'à la fin du programme, soit jusqu'à la saturation de l'écran.

Notes et commentaires

- La commande

  LIST L

  a un effet secondaire : elle déplace le *curseur de ligne en cours* sur la ligne L. Cela peut être utile quand on veut éditer une ligne particulière du programme en cours. Par exemple pour éditer la ligne 25 d'un programme, on pourrait d'abord entrer la commande :

  LIST 25

  puis taper la touche EDIT (SHIFT-1). (Pour plus d'information sur le processus d'édition, voir la rubrique EDIT.)

LLIST (shift ; touche [G])

La commande LLIST envoie le programme sur une imprimante au lieu de l'afficher sur l'écran. S'il n'y a pas d'imprimante reliée à l'ordinateur, la commande LLIST n'a aucun effet.

LN (fonction ; touche [Z])

La fonction LN donne le logarithme naturel (népérien, base e) d'un nombre. L'argument de la fonction LN doit être supérieur à 0.

Exemple de programme

La Figure L.5 montre un programme prévu pour afficher des logarithmes naturels dont les arguments varient de 90 à 0,1. Le programme comprend deux boucles FOR qui déterminent les arguments de la fonction LN. La première boucle, lignes 40 à 60, donne des arguments qui décroissent de 90 à 10. La deuxième, lignes 70 à 90, donne des arguments fractionnaires inférieurs à 1. A chaque itération, les deux boucles appellent le sous-programme de la ligne 200 pour imprimer les lignes d'information. L'expression qui contient la fonction LN se trouve à la fin de la ligne 200.

On voit sur la Figure L.6 le résultat de ce programme. Il faut

```
  10 PRINT TAB 7;"LA FONCTION ""
LN"""
  20 PRINT TAB 7;"---------------
--"
  30 PRINT
  40 FOR I=90 TO 10 STEP -10
  50 GOSUB 200
  60 NEXT I
  70 FOR I=1 TO .1 STEP -.1
  80 GOSUB 200
  90 NEXT I
 100 STOP
 200 PRINT TAB 5;"LN(";I;")";TAB
13;"= ";LN I
 210 RETURN
```

Figure L.5 : LN-Exemple de programme

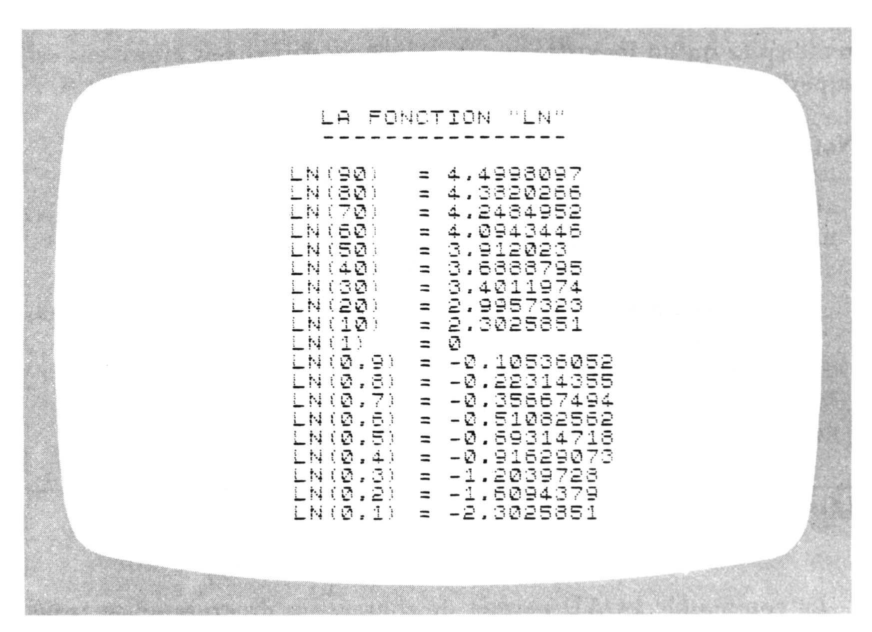


Figure L.6 : LN-Exemple de résultat

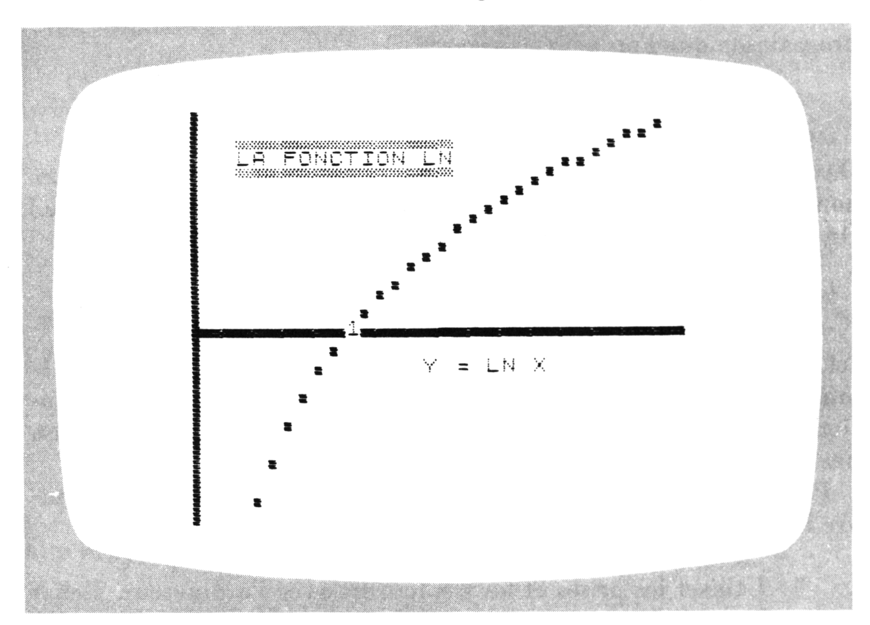


Figure L.7 : LN-Tracé de courbe

remarquer qu'un logarithme naturel est positif si son argument est supérieur à 1, et négatif si son argument est compris entre 1 et 0.

Notes et commentaires

- La Figure L.7 montre un tracé de courbe de la fonction LN. Cette courbe a été tracée à l'aide de la commande PLOT.
- Les arguments de 0 ou de valeurs inférieures à 0 sont illégaux, il en résulte un message d'erreur "A".

LOAD (mot clé ; touche [J])

La commande LOAD permet de charger un programme se trouvant sur une bande magnétique. Elle peut être exprimée sous deux formes différentes. On peut spécifier, entre guillemets, le nom du programme que l'on veut charger :

```
LOAD  "NOM"
```

Dans ce cas, l'ordinateur ne charge que le programme nommé. (Les noms doivent être *parfaitement* identiques, caractère pour caractère.) On peut également laisser le nom en blanc :

```
LOAD  ""
```

Ici, l'ordinateur charge le premier programme que lui envoie le magnétophone. (*Note* : les guillemets doubles destinés à cette utilisation sont obtenus en tapant deux fois les touches SHIFT-P et non pas les touches SHIFT-Q.)

Pour charger un programme de la bande à l'ordinateur, il faut suivre la procédure décrite ci-dessous :

1. Utiliser les prises et les fils fournis avec l'ordinateur. Relier la prise "casque/haut-parleur" du magnétophone à la prise "EAR" située sur le côté gauche de l'ordinateur.

2. Régler le volume du magnétophone à la moitié de sa puissance et la tonalité au maximum sur les aigus et sur les basses.

3. Mettre la bande dans le magnétophone puis la rembobiner, si nécessaire, jusqu'au début du programme à charger. (Si la bande contient plus d'un programme, on peut, avec un magnétophone équipé d'un compteur, noter le point de départ et les références de chaque programme.)

4. Entrer la commande LOAD (sous l'une des deux formes possibles), puis taper la touche ENTER. On voit apparaître sur l'écran de fines lignes horizontales (d'environ 6 millimètres de hauteur).

5. Appuyer sur la touche PLAY du magnétophone. Quand le programme commence à charger, on voit l'affichage de l'écran changer. Les lignes horizontales s'épaississent (environ 2,5 centimètres) et s'agitent.

6. Quand l'écran s'efface et que le message :

   ```
   0/0
   ```

   apparaît dans le coin inférieur gauche de l'écran, on sait que le programme est chargé dans l'ordinateur. Appuyer sur la touche ENTER pour visualiser le programme ou entrer la commande RUN pour commencer son exécution.

7. On peut interrompre le chargement à n'importe quel moment en tapant la touche BREAK. Si un problème semble se poser (par exemple, si le chargement du programme est trop long), taper la touche BREAK et recommencer l'ensemble du processus. Il faut s'assurer que les prises de l'ordinateur et du magnétophone sont bien connectées, et faire des essais en faisant varier le volume du magnétophone.

Notes et commentaires

- L'instruction LOAD est généralement utilisée en commande immédiate mais elle peut être insérée dans un programme.

LPRINT (shift ; touche [S])

La commande LPRINT envoie une ligne d'information à l'imprimante au lieu de l'afficher sur l'écran. S'il n'y a pas d'imprimante reliée à l'ordinateur, la commande LPRINT ne produit aucun effet.

Notes et commentaires

- L'instruction LPRINT ignore l'ordonnée d'une adresse AT tandis que l'abscisse est interprétée comme l'argument de la fonction TAB.

- Pour plus d'information sur les sorties, voir la rubrique PRINT.

Menu (vocabulaire informatique)

Le menu est l'ensemble des possibilités disponibles à un moment donné de l'exécution d'un programme. Il doit aussi indiquer une méthode claire permettant la sélection d'une de ces possibilités. (Voir la rubrique GOSUB.)

Messages d'erreur (vocabulaire informatique-ZX 81)

Après chaque exécution de programme, le ZX 81 affiche dans le coin en bas à gauche de l'écran un message qui indique l'état du programme à la fin de son déroulement. Ce message prend la forme suivante :

N/L

dans laquelle N est un chiffre ou un caractère qui représente l'un des 16 messages d'erreur, et L est le numéro de la dernière ligne de programme exécutée. Le terme "message d'erreur" n'est pas tout à fait adéquat puisque deux de ces messages signalent la fin normale d'un programme (0,9). Les messages d'erreur sont les suivants :

0 Pas d'erreur, programme terminé.

1 Non concordance entre les noms des variables de contrôle des instructions FOR et NEXT.

2 Nom de variable non définie.

3 Indice de tableau en dehors des limites spécifiées dans l'instruction DIM.

4 Saturation de l'espace mémoire.

5 Saturation d'écran.

6 Résultat de calcul trop grand pour être manipulé par l'ordinateur.

7 Instruction RETURN non précédée de l'instruction GOSUB.

8 Instruction INPUT utilisée en commande immédiate.

9 Arrêt de programme à une instruction STOP.

A Utilisation illégale d'une fonction (par exemple : SQR -1 ou LN 0).

B Adresse, code ou nombre entier illégaux dans des instructions telles que PRINT AT, PLOT, CHR$,...

C L'instruction VAL ne peut pas convertir la chaîne de caractères en nombre.

D Programme interrompu par les touches BREAK ou STOP.

E Non utilisé.

F Commande de sauvegarde (SAVE) du programme sans en avoir donné le nom.

NEW (mot clé ; touche [A])

La commande NEW efface complètement de la mémoire de l'ordinateur le programme en cours. Après l'entrée de la commande NEW, il n'y a aucun moyen de retrouver le programme sauf si celui-ci a été stocké auparavant sur une bande magnétique.

NEXT (mot clé ; touche [N])

L'instruction NEXT marque la fin d'une séquence de lignes de programme qui forme une bouche FOR. Elle prend la forme suivante :

```
NEXT V
```

dans laquelle V est la variable de contrôle établie dans l'instruction FOR qui introduit la boucle. (Voir la rubrique FOR.)

NOT (fonction ; touche [N])

La fonction logique NOT inverse une expression logique dans une instruction IF.

- Si l'expression logique est "vraie", la fonction NOT la rend "fausse".

• Si l'expression logique est "fausse", la fonction NOT la rend "vraie".

La fonction NOT doit toujours apparaître immédiatement avant l'expression qu'elle modifie :

IF NOT (expression logique) **THEN** (instruction)

Exemple de programme

La portion du programme de la FigureN.1 illustre l'emploi de la fonction NOT. Le but de ces lignes est d'obtenir de l'utilisateur du programme une réponse affirmative ou négative à la question :

VOULEZ-VOUS VOIR UN RAPPORT?

les mots :

O)UI OU N)ON

apparaissent dans le coin inférieur gauche de l'écran pour indiquer à l'utilisateur les deux réponses possibles à cette question, soit "O", soit "N".

Dans une situation comme celle-ci, on devrait toujours, au moment où l'on écrit les instructions, envisager une erreur éventuelle de la part de l'utilisateur qui peut accidentellement taper une mauvaise touche. On imagine par exemple que l'utilisateur veuille visualiser un rapport, et qu'il entre par inadvertance la réponse "T" au lieu de "O". Si le programme possède une instruction telle que :

IF A$< >"O" **THEN STOP**

pour tester la valeur de la variable "réponse" A$, l'utilisateur n'aura pas la possibilité de corriger ce qui n'est qu'une simple erreur de frappe.

Les lignes de la Figure N.1 sont conçues pour éviter ce désagrément. Les lignes 10 et 20 impriment la question ainsi que les mots O)UI OU N)ON. Puis la ligne 50 lit ce que l'utilisateur a entré au clavier et l'assigne à la variable A$. La ligne 60, qui montre un exemple de fonction NOT, teste la valeur de A$:

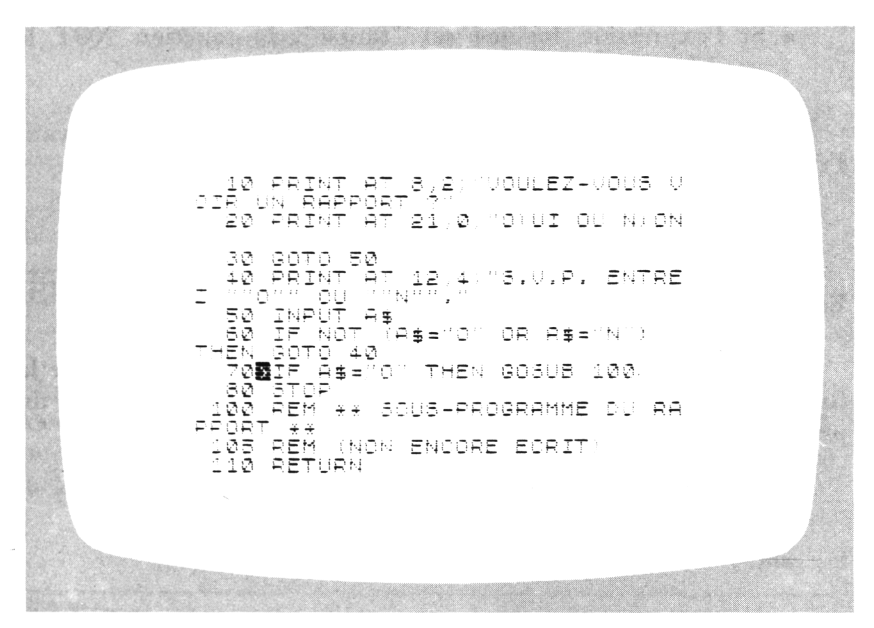


Figure N.1 : NOT-Exemple de programme

60 **IF NOT** (A$="O" OR A$="N") **THEN GOTO** 40

Cette ligne permet de corriger d'éventuelles erreurs de frappe. Si A$ contient l'une des valeurs adéquates, "O" ou "N", la ligne 60 n'a aucune conséquence. Mais si A$ contient une valeur autre que "O" ou "N" (par exemple "T"), elle renvoie le programme à la ligne 40. Cette ligne émet un message *supplémentaire* pour indiquer à l'utilisateur qu'il s'est trompé. La ligne 50 est ensuite exécutée à nouveau pour lire la nouvelle valeur donnée à A$.

Une fois que la ligne 60 a testé la réponse avec satisfaction, la ligne 70 renvoie le programme à la ligne de sous-programme 100 si l'utilisateur veut voir le rapport :

70 **IF** A$="O" **THEN GOSUB** 100

En fait, les lignes 100 à 110 ne sont que le "talon" d'un sous-programme qu'il reste à écrire. D'autre part, il faut éclaircir le point

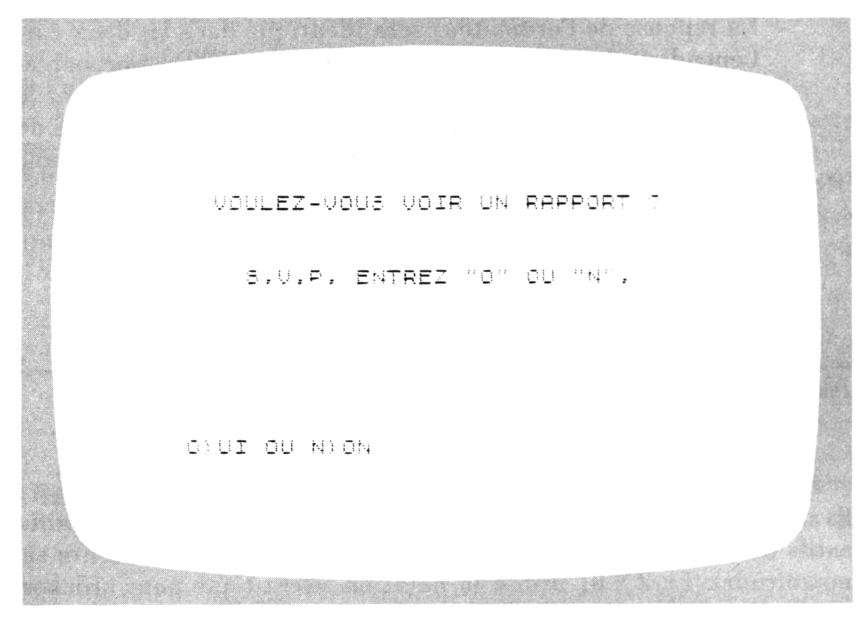


Figure N.2 : NOT-Exemple de résultat

suivant : l'instruction IF, ligne 60, facilite l'utilisation du programme en prévoyant des réponses erronées et des moyens pour les corriger.

La Figure N.2 montre ce programme. L'apparition du deuxième signal réclamant l'entrée d'une réponse (S.V.P.ENTREZ "O" OU "N") indique que l'utilisateur a déjà fait au moins une erreur.

Notes et commentaires

- Les instructions IF qui contiennent la fonction NOT peuvent toujours être réécrites de façon à éliminer NOT. Par exemple, la ligne 60 aurait pu être écrite de la manière suivante :

```
60 IF A$<>"O" AND A$<>"N" THEN GOTO 40
```

La réponse de l'ordinateur sera identique dans les deux cas. Cependant, la version utilisant la fonction NOT est probablement plus facile à comprendre pour une personne qui lit le programme pour la première fois. Dans ce cas, l'utilisation de la fonction NOT est motivée par le désir de programmer dans un style clair.

Notation scientifique (vocabulaire informatique)

La notation scientifique est une méthode d'écriture de nombres : ils sont divisés en deux parties distinctes, la *mantisse* (chiffres significatifs du nombre) et l'*exposant* (puissance de dix qui représente sa magnitude). Le ZX 81 utilise la notation scientifique pour afficher sur l'écran des nombres très grands ou très petits. Par exemple :

```
2.34E+14
```

Dans ce nombre, la valeur située avant la lettre E est la mantisse et la valeur située après la lettre E est la puissance de dix. On peut lire ce nombre de la manière suivante "2,34 fois dix puissance 14" :

```
2,34*100 000 000 000 000
```

ou

```
234 000 000 000 000
```

Un exposant négatif transforme le nombre en une valeur fractionnaire. Par exemple,

```
8.7E-10
```

signifie :

```
0,00000000087
```

Octet (Byte) (vocabulaire informatique)

Un octet est une unité d'espace mémoire ; c'est la place nécessaire au stockage d'un caractère en mémoire. Le ZX 81 a 1 K octet (environ 1000 octets) de mémoire utilisable pour un programme BASIC et pour les variables qui lui sont nécessaires. On peut lui adjoindre une extension portant la capacité mémoire à 16 K (environ 16 000 octets).

OR (shift ; touche [W])

L'opérateur logique OR (OU) peut être utilisé dans la création d'une expression logique pour une décision IF. La valeur d'une telle expression dépend des valeurs des éléments combinés. On prend une expression de la forme suivante :

proposition-1 **OR** proposition-2

L'expression composée est vraie si l'une des deux propositions est vraie ou si les deux sont vraies. Si les deux sont fausses, l'expression composée est fausse.

Exemple de programme

L'exemple de programme décrit dans la rubrique AND (Figure A.3) contient également un exemple d'utilisation de l'opérateur OR. La ligne 180 de ce programme teste les valeurs des deux variables MOY ("moyenne des résultats des questionnaires") et F ("résultat de l'examen") :

```
18 IF MOY<75 OR F<70
THEN PRINT AT 9,5;"**RECALE"
```

Si l'un des deux résultats est inférieur au minimum requis (75 pour MOY, 70 pour F), l'expression composée :

MOY<75 **OR** F<70

est estimée "vraie", ce qui entraîne l'affichage du message "**RE-CALE". L'expression composée est estimée "fausse" seulement si les deux éléments qui la composent sont faux, c'est-à-dire si les deux résultats sont égaux ou supérieurs au minimum requis.

Notes et commentaires

- La Figure O.1 est une "table de vérité" pour les conditions OR. Elle montre le résultat d'une expression composée, en fonction des différentes combinaisons possibles des valeurs des propositions 1 et 2. On remarque que l'expression composée est vraie dans trois cas : quand l'une ou l'autre, ou les deux propositions sont vraies.

OR
====

TABLE DE VERITE

PROPOS. 1	PROPOS. 2	PROPOS. COMPOSEE
VRAIE	VRAIE	VRAIE
VRAIE	FAUSSE	VRAIE
FAUSSE	VRAIE	VRAIE
FAUSSE	FAUSSE	FAUSSE

Figure O.1 : OR-Table de vérité

- Pour plus d'information, voir les rubriques AND, IF, NOT et THEN.

PAUSE (mot clé ; touche [M])

L'instruction PAUSE commande à l'ordinateur d'arrêter momentanément l'exécution d'un programme. Elle prend la forme :

```
PAUSE N
```

dans laquelle N est un nombre compris entre 0 et 65535. La valeur de N détermine la durée de la pause. Une incrémentation de N de 60 correspond à une augmentation de la durée de la pause d'environ une seconde. En conséquence, la commande :

```
PAUSE 60*S
```

entraîne un arrêt d'environ S secondes.

Si le nombre suivant l'instruction est supérieur à 32767, l'ordinateur ne chronomètre pas la pause. Celle-ci a une durée illimitée, jusqu'à ce qu'une touche quelconque du clavier soit tapée.

Exemple de programme

Le programme de la Figure P.1 montre deux utilisations différentes de l'instruction PAUSE. Ce programme présente une sorte de test de vitesse de lecture. Il affiche puis efface presque simultanément les mots d'une phrase, en mettant l'utilisateur au défi de lire la phrase en un temps très court.

La ligne 10 attribue les mots de la phrase à la chaîne de caractères S$. La phrase est complétée par des espaces de façon que chaque mot contienne exactement sept caractères. Les lignes 20 et 30 affichent sur l'écran le titre et les instructions, ainsi que le message :

```
POUR COMMENCER TAPEZ UNE TOUCHE
```

```
    5 SLOW
    7 LET T=5
   10 LET S$=" SI VOUS POUVEZ LIR
E A CETTE VITESSE BRAVO"
   20 PRINT AT 8,4;"TEST DE VITES
SE PRET"
   30 PRINT AT 10,5;"TAPEZ UNE TO
UCHE"
   40 PAUSE 40000
   50 CLS
   60 FOR I=1 TO 6
   70 PRINT AT 8,12;S$(I*7-6 TO I
*7)
   80 PAUSE T
   90 PRINT AT 8,12;"       "
  100 PAUSE T/2
  110 NEXT I
```

Figure P.1 : PAUSE-Exemple de programme

Ceci est possible grâce à l'instruction PAUSE de la ligne 40 :

40 **PAUSE** 40000

Cette instruction provoque une pause d'une durée indéterminée qui peut être interrompue par la frappe de n'importe quelle touche. A ce moment-là, le programme repart.

Les mots de la phrase-test sont à nouveau affichés puis effacés un par un par l'intermédiaire de la boucle FOR (ligne 60 à 110). La ligne 70 utilise le processus de « découpage » de chaîne pour afficher des portions de la chaîne S$ comportant sept caractères (pour plus d'information sur le « découpage », voir la rubrique TO). La ligne 90 efface par la suite ces mêmes sept caractères. Chacune de ces lignes est suivie d'une instruction PAUSE qui ralentit le déroulement du test :

80 **PAUSE** T
100 **PAUSE** T/2

On assigne la valeur 5 à la variable T au début du programme. On peut faire varier la vitesse du test de lecture en diminuant ou en augmentant la valeur de T.

PEEK (fonction ; touche [0])

La fonction PEEK donne le contenu d'une adresse mémoire déterminée sous forme décimale. L'instruction PEEK se présente de la façon suivante :

```
PEEK M
```

dans laquelle M est une valeur numérique littérale, une variable ou une expression arithmétique qui représente une adresse mémoire comprise entre 0 et 65535. La fonction PEEK donne un nombre décimal compris entre 0 et 255 qui représente le contenu d'un octet de mémoire.

Les instructions PEEK et POKE, qui à elles deux permettent un accès à l'organisation interne de l'ordinateur beaucoup plus direct que celui offert par les autres commandes BASIC, sont utiles aux programmeurs qui veulent élaborer des sous-programmes en *code machine*. Ces sous-programmes demandent une connaissance du jeu d'instructions en code machine du Z80, qui est le microprocesseur du ZX 81. (Pour plus d'information, voir les rubriques Code Machine, POKE et USR.)

Exemple de programme

Le programme de la Figure P.2 illustre l'utilisation de la fonction PEEK. C'est un exercice destiné à localiser les adresses mémoire dans lesquelles ce programme est stocké, et à afficher leur contenu sur l'écran.

Le programme commence par une ligne REM qui sert à signaler le début du programme en mémoire :

```
10 REM LOCALISER CETTE PHRASE DANS LA MEMOIRE
   DE L'ORDINATEUR
```

```
   10 REM LOCALISER CETTE PHRASE
DANS LA MEMOIRE DE L'ORDINATEUR.
   15 REM
   20 FOR I=16500 TO 16700
   30 IF INT (I/10)*10<>I THEN GO
TO 60
   40 PRINT
   50 PRINT " ";I;"> ";
   60 PRINT CHR$ PEEK I;
   70 NEXT I
   80 STOP
```

Figure P.2 : PEEK-Exemple de programme

```
16500>  ■■
16510>  ▒T  REM LOCALI
16520>  SER CETTE
16530>  PHRASE DAN
16540>  S LA MEMOI
16550>  RE DE L OR
16560>  DINATEUR.?
16570>   ?"  REM
16580>      ? =.  FOR
16590>  I=16500?■
16600>   CONT   TO 16700?
16610>  ■■?  ? 28
16620>   IF INT (I/10?■4
16630>      )+10?■4
16640>      <>I THEN GOTO 60?
16650>  ■?   ? C■
16660>   PRINT ?'M$  PRINT " "
16670>  ;I;"> ";?
16680>  W■  PRINT CHR$ PEEK I;?
16690>  ?■  NEXT I? ?■
16700>   STOP
```

Figure P.3 : PEEK-Exemple de résultat

Les lignes 20 à 70 forment une boucle FOR qui lit les adresses mémoire 16500 à 16700 et affiche leur contenu sur l'écran :

```
20 FOR I=16500 TO 16700
```

Le programme se trouve dans cette zone mémoire. Les lignes 30 à 50 affichent chaque adresse multiple de 10. La ligne 50 affiche le nombre I :

```
50 PRINT " ";I;"> ";
```

et la ligne 30 saute à l'instruction PRINT à la ligne 50 pour toutes les valeurs de I qui ne sont pas des multiples de 10 :

```
30 IF INT(I/10)*10<>I THEN GOTO 60
```

La ligne 60 affiche le contenu des adresses mémoire. Il ne faut pas oublier que la fonction PEEK donne ce contenu sous forme décimale. Par conséquent, la ligne 60 utilise la fonction CHR$ pour convertir chaque valeur dans le caractère qui lui correspond en code caractère du ZX 81 :

```
60 PRINT CHR$ PEEK I;
```

La Figure P.3 montre le résultat de ce programme. On voit que la première instruction REM commence à l'adresse 16513.

PI (fonction ; touche [M])

La fonction PI donne la valeur de π, quelle que soit l'expression arithmétique dans laquelle on l'utilise. Alors que la fonction apparaît en lettre grecque sur le clavier, elle est affichée sur l'écran sous la forme PI.

Pour obtenir la valeur de PI, entrer la commande suivante :

```
PRINT PI
```

Le résultat affiché sur l'écran est :

```
3.1415927
```

Le programme de la rubrique PLOT ainsi que ceux des rubriques COS, SIN et TAN (Figures P.5, C.14, S.5 et T.3) montrent de bons exemples d'utilisation de la fonction PI.

PLOT (mot clé ; touche [Q])

La commande PLOT affiche un *point de graphisme* (pixel, de *picture element)* sur l'écran à une adresse déterminée. La forme courante de l'instruction PLOT est :

```
PLOT H,V
```

dans laquelle H et V sont les coordonnées horizontales et verticales de l'adresse PLOT. H et V peuvent être exprimées comme des valeurs numériques littérales, des variables ou des expressions arithmétiques.

Pour voir à quoi ressemble un point de graphisme, on peut entrer l'exemple de commande PLOT suivant :

```
PLOT 31,21
```

Elle affichera un point approximativement au centre de l'écran. Pour comprendre le fonctionnement des adresses PLOT, il faut imaginer un écran divisé en 64 colonnes et 44 rangées. Chaque petit carré formé par l'intersection d'une colonne et d'une rangée a la taille d'un point graphique. L'adresse d'un point graphique situé dans le coin en bas à gauche de l'écran est 0,0. Les coordonnées horizontales de l'adresse PLOT varient de 0 à 63 quand on déplace le point de gauche à droite sur l'écran. Les coordonnées verticales varient de 0 à 43 quand on déplace le point du bas vers le haut sur l'écran. L'utilisation de la commande PLOT (ainsi que celle de la commande inverse,

UNPLOT) est résumée dans la Figure P.4. Cette figure montre également les adresses des points situés aux quatre coins de l'écran.

Exemple de programme

La Figure P.5 montre un exercice graphique simple qui dessine des cercles sur l'écran. Le programme est destiné à lire à partir du clavier les informations qui définissent chaque cercle. On peut créer des cercles de n'importe quelle dimension, à n'importe quel endroit sur l'écran.

Les lignes 20 à 80 permettent l'entrée des données. Le rayon (exprimé en points graphiques) est stocké dans la variable R. Les coordonnées horizontales et verticales du centre du cercle sont stockées respectivement dans les variables CH et CV.

Ensuite, la boucle FOR (lignes 90 à 130) dessine le cercle. Le programme utilise les fonctions SIN et COS pour calculer les coordonnées du cercle, ainsi la boucle FOR incrémente l'angle A de 0 à 2 :

```
90 FOR A=0 TO 2*PI STEP PI/R
```

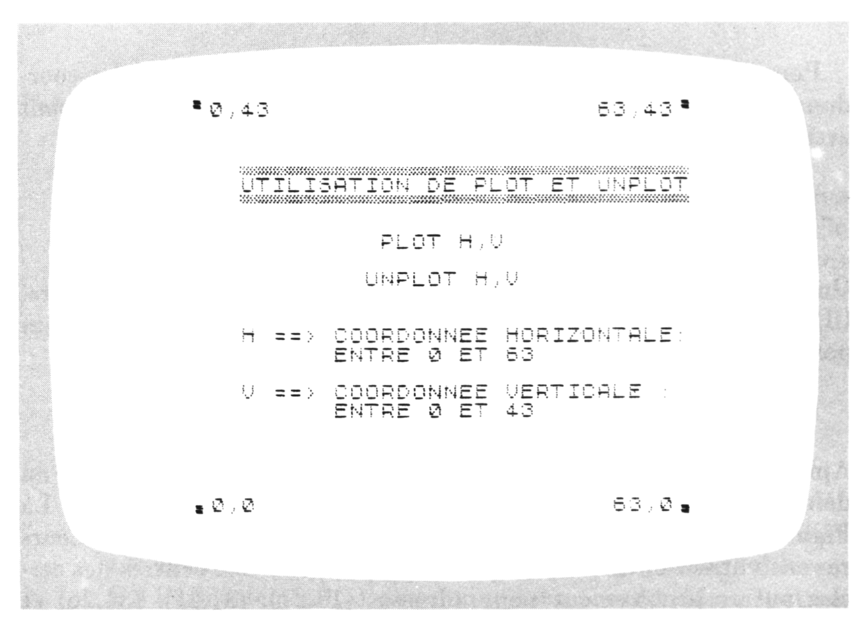


Figure P.4 : PLOT-Résumé

```
 130 PRINT AT 0,0;"▪0,43"
 140 PRINT AT 0,26;"63,43▪"
 150 PRINT AT 21,0;"▪0,0 "
 160 PRINT AT 21,26;" 63,0▪"
 165 PRINT AT 3,3;"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
 170 PRINT AT 4,3;"UTILISATION D
E PLOT ET UNPLOT"
 175 PRINT AT 5,3;"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
 180 PRINT
 190 PRINT TAB 12;"PLOT H,V"
 200 PRINT
 210 PRINT TAB 11;"UNPLOT H,V"
 220 PRINT
 230 PRINT
 240 PRINT TAB 3;"H ==> COORDONN
EE HORIZONTALE:"
 250 PRINT TAB 9;"ENTRE 0 ET 63"
 260 PRINT
 270 PRINT TAB 3;"V ==> COORDONN
EE VERTICALE :"
 280 PRINT TAB 9;"ENTRE 0 ET 43"
 290 INPUT X
```

Figure P.5 : PLOT-Exemple de programme

Pour chaque angle A, deux instructions LET calculent les coordonnées x et y d'un point situé sur la circonférence. Ces valeurs sont attribuées aux variables X et Y :

100 **LET** X = R ***COS** A + CH
110 **LET** Y = R ***SIN** A + CV

On remarque que ces calculs nécessitent les coordonnées du centre, CH et CV. Pour finir, la commande PLOT (ligne 120) affiche un point graphique pour chaque point du cercle calculé :

120 **PLOT** X,Y

Après le tracé d'un cercle, la ligne 140 renvoie le programme au début (pour l'entrée des coordonnées d'un nouveau cercle). La Figure P.6 montre quatre cercles dessinés par ce programme. Leurs rayons varient entre 5 et 10 (points graphiques). Les centres des cercles ont respectivement pour adresses (15,21), (31,21), (31,26) et (50,21).

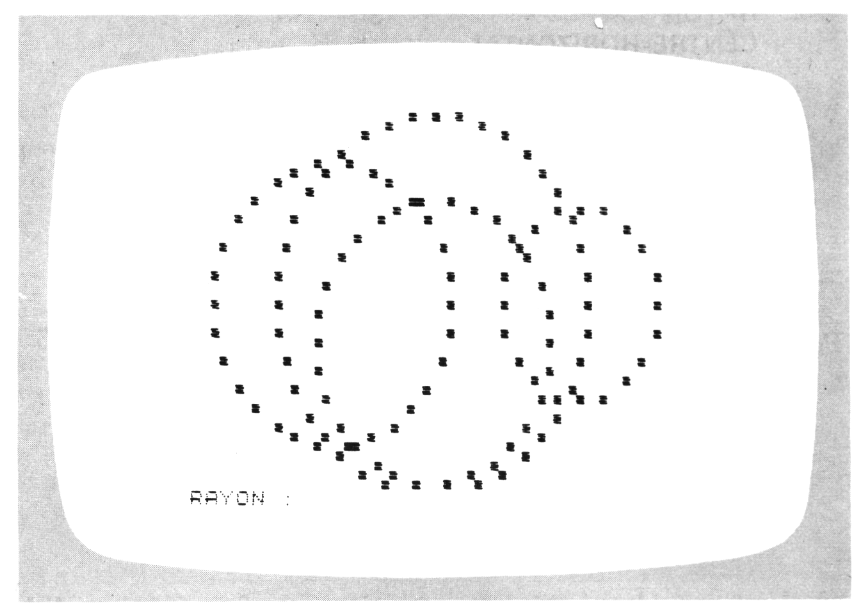


Figure P.6 : PLOT-Exemple de résulat

Notes et commentaires

- Si l'on donne une valeur négative à une coordonnée de l'instruction PLOT, l'ordinateur prend la valeur absolue de celle-ci. Si cette valeur absolue fait partie des coordonnées d'adresses légales de PLOT, un point graphique apparaît sur l'écran. Cela signifie que les coordonnées horizontales et verticales, H et V, sont dans les limites suivantes :

  -63 < = H < = +63
  -43 < = V < = +43

  (Une adresse contenant une coordonnée en dehors de ces limites entraîne la fin du programme avec un message d'erreur B.) Parfois le résultat obtenu avec des coordonnées négatives peut surprendre. Par exemple, si l'on refait tourner le programme en définissant le cercle de la manière suivante :

RAYON	30
CENTRE-HORIZONTAL	5
CENTRE-VERTICAL	3

Peut-on voir quelle partie du dessin est produite par des coordonnées négatives ?

POKE (mot clé ; touche [O])

La commande POKE écrit une valeur à une adresse déterminée dans la mémoire. L'instruction POKE s'écrit de la façon suivante :

POKE M,V

où M est une adresse mémoire comprise entre 0 et 65535, et V la valeur à écrire à cette adresse. Cette valeur doit satisfaire la relation :

-255 < = V < = +255

M et V peuvent être toutes deux exprimées sous forme de valeurs numériques littérales, de variables ou d'expressions arithmétiques.

L'instruction POKE peut être utilisée pour stocker des instructions en code machine du Z80 dans la mémoire de l'ordinateur. Les instructions du microprocesseur Z80 (unité centrale du ZX 81) sont toutes codées de 0 à 255. La possibilité d'écrire une séquence d'instructions en code machine, qui exécutera une certaine tâche à un point donné d'un programme BASIC, peut parfois être utile. Cependant, cette opération nécessite une connaissance approfondie du jeu d'instructions du Z80.

Exemple de programme

Le programme de la Figure P.7 fait une brève démonstration des possibilités offertes par l'instruction POKE. Ce programme est une extension du programme décrit dans la rubrique PEEK (Figure



Figure P.7 : POKE-Exemple de programme

P.2). Le programme principal, lignes 10 à 80, se contente de lire les instructions après leur stockage dans la mémoire de l'ordinateur. En faisant tourner une première fois cette partie du programme (voir la rubrique PEEK), on s'est aperçu que le premier mot clé, REM, est stocké en mémoire à l'adresse 16513. Il y a maintenant un sous-programme supplémentaire, ligne 100, qui utilise l'instruction POKE pour *modifier les valeurs* de plusieurs adresses mémoire au début du programme.

La ligne 15 appelle le sous-programme avant l'affichage du contenu de la zone mémoire. Le sous-programme commence par deux instructions d'affectation :

```
100 LET M=16513
110 LET W$="MODIFIER "
```

La variable M contient l'adresse mémoire du début du programme. La chaîne de caractères W$ contient les caractères que le sous-programme doit écrire dans la mémoire de l'ordinateur, immédiatement après l'adresse 16513.

La boucle FOR de la ligne 120 incrémente la variable de contrôle I de 1 à la longueur (en caractères) de la chaîne W$:

```
120 FOR I=1 TO LEN W$
```

La boucle contient une seule instruction, l'instruction POKE :

```
130 POKE M+I,CODE W$(I)
```

A partir de l'adresse 16514 (M + I), chaque caractère de W$ est converti en son équivalent décimal (par l'intermédiaire de la fonction CODE) et stocké dans la mémoire de l'ordinateur. Le travail du sous-programme s'arrête là.

La Figure P.8 montre le résultat du programme. Si l'on compare ces résultats avec ceux de la Figure P.3, on voit que le mot "LOCALISER" (adresses 16514 à 16522) a été remplacé par le mot "MODIFIER". C'est le résultat de l'instruction POKE du sous-programme.

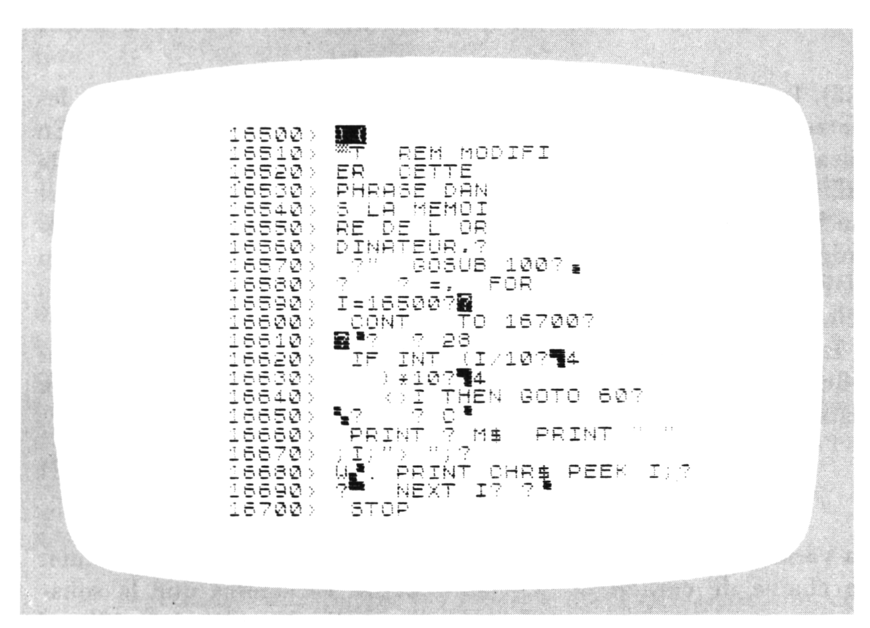


Figure P.8 : POKE-Exemple de résultat

```
  10 REM MODIFIER  CETTE PHRASE
DANS LA MEMOIRE DE L ORDINATEUR.
  15 GOSUB 100
  20 FOR I=16500 TO 16700
  30 IF INT (I/10)*10<>I THEN GO
TO 60
  40 PRINT
  50 PRINT "  ";I;"> ";
  60 PRINT CHR$ PEEK I;
  70 NEXT I
  80 STOP
 100 LET M=16513
 110 LET W$="MODIFIER "
 120 FOR I=1 TO LEN W$
 130 POKE M+I,CODE W$(I)
 140 NEXT I
 150 RETURN
```

Figure P.9 : POKE-Programme modifié

Pour finir, et après déroulement complet, le programme est lui aussi différent (voir Figure P.9). La ligne REM dit maintenant "MODIFIER CETTE PHRASE..." au lieu de "LOCALISER CETTE PHRASE...".

Il faut remarquer que les utilisateurs du ZX 81 emploient souvent les lignes REM pour stocker les instructions en code machine. En utilisant ce moyen, on évite les interférences avec le programme BASIC. La technique de programmation illustrée par cet exemple (utilisation de l'instruction POKE pour mettre des valeurs dans une ligne REM placée au début d'un programme) est celle que l'on utilise pour stocker un programme en code machine dans la mémoire de l'ordinateur.

PRINT (mot clé ; touche [P])

La commande PRINT affiche des informations sur l'écran. Une seule instruction PRINT peut provoquer l'affichage de nombreux

éléments : valeurs littérales (nombres et chaînes de caractères), valeurs de variables numériques ou de variables de chaînes et même résultats d'expressions numériques, logiques ou de chaînes. De plus, l'instruction PRINT offre de nombreuses possibilités d'affichage de données et de caractères graphiques à n'importe quel endroit sur l'écran.

Les Figures P.10 et P.11 donnent huit exemples d'utilisation de l'instruction PRINT et l'affichage de leurs résultats sur l'écran. Ces exemples illustrent la variété des possibilités offertes par l'instruction PRINT. Les notes qui suivent commentent chacun des huit exemples :

1. **Utilisation du point-virgule.** Deux éléments d'une instruction PRINT séparés par un point-virgule sont affichés côte à côte sur l'écran, sans espace les séparant, comme le montre l'exemple suivant. De plus, quand une instruction PRINT *se termine* par un point-virgule, l'instruction PRINT suivante commence l'affichage là où le précédent affichage s'est arrêté. On considère par exemple les lignes suivantes :

```
 5 LET J$="MARCEL" 10 PRINT "BONJOUR ";
20 IF J$<>"" THEN PRINT J$
```

   Le point-virgule situé à la fin de la ligne 10 empêche l'ordinateur de déplacer l'affichage sur une nouvelle ligne lors de la commande PRINT suivante. Le résultat de cette séquence est donc :

```
BONJOUR MARCEL
```

2. **Utilisation de la virgule.** Une virgule qui sépare deux éléments d'une instruction PRINT provoque une tabulation qui est liée à la position de l'affichage précédent (soit au milieu de la ligne en cours, soit au début de la ligne suivante). Ainsi, dans l'exemple de la Figure P.10, la première virgule place Y au milieu de la ligne sur laquelle X est affiché. La seconde virgule place Z au début de la ligne suivante.

3. **Utilisation de la fonction TAB.** L'argument de la fonction TAB indique, sur la ligne d'affichage en cours, le numéro de

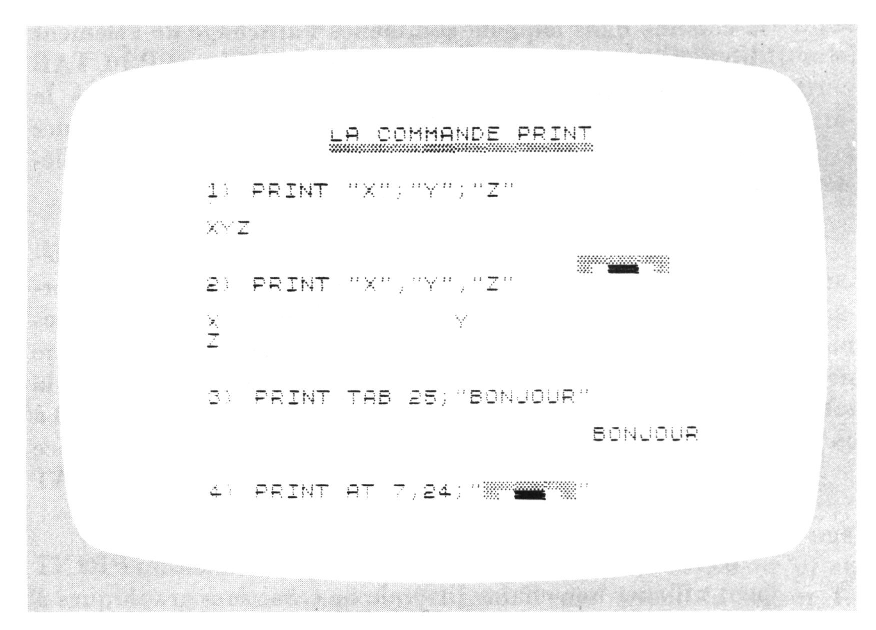


Figure P.10 : PRINT-Exemples

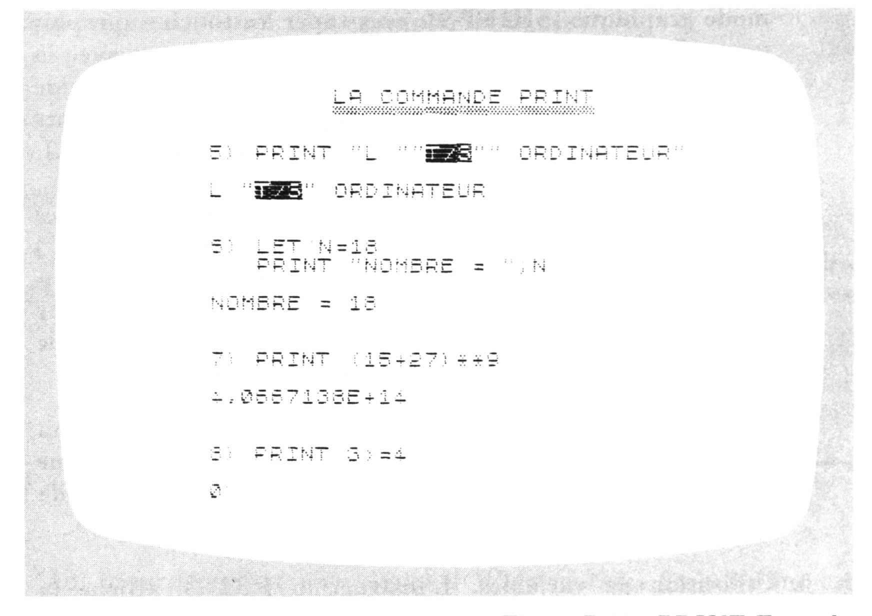


Figure P.11 : PRINT-Exemples

la colonne dans laquelle commence l'affichage de l'élément suivant l'instruction. Dans l'exemple de la Figure P.10, TAB 27 signifie que le "B" de BONJOUR apparaît dans la colonne 27, suivi du reste des caractères. Il faut remarquer qu'un point-virgule doit séparer la fonction TAB de l'élément à afficher (voir la rubrique TAB).

4. **Utilisation de la fonction AT**. La fonction AT affiche un élément à une adresse déterminée sur l'écran. Les deux coordonnées d'adresse de l'instruction AT doivent être séparées de l'élément à afficher par un point-virgule. La première coordonnée représente le numéro de rangée (de 0 à 21), la seconde coordonnée représente le numéro de colonne (de 0 à 31). Dans l'exemple de la Figure P.10, l'affichage commence rangée 7, colonne 24. Un point-virgule sépare la fonction AT de l'élément à afficher. (Voir la rubrique AT.)

   Utilisation des caractères graphiques. L'instruction PRINT peut afficher une chaîne littérale de caractères graphiques à n'importe quel endroit sur l'écran. Pour pouvoir écrire des caractères graphiques, il faut d'abord mettre le clavier en mode graphique (SHIFT-9), puis taper les touches (graphiques). Tous les caractères graphiques sont obtenus avec la touche SHIFT. La chaîne de cet exemple est obtenue à l'aide des caractères graphiques situés sur la séquence de touches suivante : A, S, F, F, S, A. (Voir la rubrique GRAPHICS.)

5. **Utilisation des doubles guillemets** (""). La touche doubles guillemets (SHIFT-Q) produit des guillemets simples (") lors de l'affichage d'une chaîne littérale. Il ne faut pas confondre les doubles guillemets avec les vrais guillemets (SHIFT-P) qui sont utilisés pour délimiter les chaînes de caractères.

   Utilisation des caractères en vidéo inverse. L'instruction PRINT peut afficher des caractères en vidéo inverse. Pour écrire ces caractères, il faut d'abord placer le clavier en mode graphique (SHIFT-9) puis taper n'importe quelle touche.

6. **Utilisation de variables.** L'instruction PRINT affiche la *valeur* d'une variable qui lui est associée. Dans l'exemple de

la Figure P.11, la valeur 18, assignée à la variable numérique N, est affichée à la suite d'une valeur de chaîne littérale. Pour séparer par des espaces deux éléments d'une instruction PRINT, il faut ajouter ces espaces dans la chaîne littérale. (Voir dans cet exemple les espaces situés de part et d'autre du signe égal de la chaîne ainsi que l'affichage correspondant.)

7. **Expressions arithmétiques et notation scientifique.** Lorsqu'une expression arithmétique fait partie d'une instruction PRINT, l'ordinateur évalue d'abord l'expression puis affiche le résultat sur l'écran. Celui-ci est affiché en notation scientifique s'il est supérieur ou égal à 10^{13}, ou inférieur à 10^{-5}. (Voir les rubriques Expression arithmétique et Notation scientifique.)

8. **Expressions logiques.** Dans une instruction PRINT, une expression logique estimée "fausse" prend la valeur 0, et une expression logique estimée "vraie" prend la valeur 1.

Note : l'instruction PRINT seule n'affiche rien ; il en résulte une ligne vide sur l'écran. Par exemple, les lignes :

```
10 PRINT "QUELQUE CHOSE"
20 PRINT
30 PRINT "QUELQUE CHOSE D'AUTRE"
```

entraînent l'affichage d'une ligne vide entre les deux résultats.

Presque tous les programmes de ce livre contiennent des exemples d'utilisations de la commande PRINT. On comprendra mieux ces exemples en étudiant les Figures P.10 et P.11.

Programme (vocabulaire informatique)

Un programme est une séquence d'instructions écrites dans un langage compris par l'ordinateur et destinées à lui faire accomplir

une tâche définie. Les lignes d'instructions d'un programme BASIC sont numérotées. L'ordinateur garde simplement les instructions en mémoire jusqu'à ce qu'on lui donne l'ordre de commencer l'exécution du programme. L'affichage des lignes d'un programme (soit sur écran, soit sur papier) est appelé *listing*.

Programme d'aide à l'utilisateur (vocabulaire informatique)

Ce programme est destiné à aider la personne qui l'utilise. Voici quelques-uns des éléments qui le composent :

- une description claire des possibilités offertes à l'utilisateur et une méthode pour faire un choix parmi ces possibilités,
- des signaux clairs et précis, qui indiquent à l'utilisateur quelles sortes de données l'ordinateur attend du clavier, et le moment auquel il doit les introduire,
- des moyens efficaces et "encourageants" pour réparer des erreurs d'entrée de données.

Pour plus d'information et de commentaires, voir les rubriques GOSUB, INKEY$, INPUT, NOT et VAL.

Programmeur (vocabulaire informatique)

Le programmeur est la personne qui écrit un programme, par opposition à *l'utilisateur* qui fait exécuter le programme et dialogue avec lui. Dans le cas d'un ordinateur individuel, le programmeur et l'utilisateur sont souvent une seule et même personne.

RAND (mot clé ; touche [T])

L'instruction RAND introduit la "valeur de départ" de la fonction RND, et détermine donc le point de départ d'une série de nombres aléatoires générés par cette fonction.

La fonction RND donne des nombres qui semblent aléatoires pour la plupart des applications. Ces nombres sont en fait le résultat d'une formule dans laquelle la valeur de départ détermine le point de départ de cette fonction. Dans certaines situations, on peut avoir besoin de demander à l'ordinateur de reproduire une même séquence de nombres aléatoires ; par exemple, pour réexaminer un jeu qui utilise des nombres aléatoires ou pour reproduire le scénario d'une simulation. Dans ces deux exemples, on peut avoir besoin de faire tourner un programme plusieurs fois en ayant l'assurance qu'à chaque fois l'ordinateur générera la même séquence de nombres aléatoires. Ceci est réalisable avec l'instruction RAND.

Par ailleurs, on peut avoir besoin d'une série de nombres aléatoires totalement imprévisibles (pour un jeu, une simulation ou pour n'importe quel autre programme qui utilise la fonction RND). La commande RAND permet également de générer une telle série.

L'instruction RAND prend une des deux formes suivantes :

RAND

ou **RAND N**

dans laquelle N est un nombre compris entre 0 et 65536. L'instruction RAND seule donne le même résultat que l'instruction RAND 0. Ces deux instructions génèrent un point de départ imprévisible pour la fonction RND. N'importe quelle autre instruction RAND dans laquelle N est supérieur à 0 produit une séquence de nombres aléatoires spécifique et reproductible.

Exemple de programme

Le programme de la Figure R.1 et l'affichage de résultat qui l'accompagne (Figure R.2) montrent clairement ce qui différencie l'instruction RAND 0 de l'instruction RAND N (dans laquelle N est différent de 0). Le programme génère et affiche deux séries de nom-

```
  10 PRINT TAB 7;"L INSTRUCTION
 ""RAND"" "
  20 FOR S=0 TO 1
  30 PRINT AT (S*11)+2,0;"SERIES
 NO. ";S+1
  40 FOR R=0 TO 2
  50 RAND R*10
  60 PRINT AT (S*11)+4,R*11;"RAN
D ";R*10
  70 FOR N=6 TO 10
  80 PRINT AT (S*11)+N,R*11;RND
  90 NEXT N
 100 NEXT R
 110 NEXT S
```

Figure R.1 : RAND-Exemple de programme

```
       L INSTRUCTION  "RAND"

SERIES NO. 1

RAND 0     RAND 10    RAND 20

0,14602661 .012573242 .024017334
0,95297241 0,94412232 0.80241394
0,47297669 0,80923462 0.18125916
0,47384644 0,69281006 0.59536743
0,53907776 0,96110535 0.65301514

SERIES NO. 2

RAND 0     RAND 10    RAND 20

0,53950501 .012573242 .024017334
0,46339417 0,94412232 0.80241394
0,75517273 0,80923462 0.18125916
0,63822937 0,69281006 0.59536743
0,86761475 0,96110535 0.65301514
```

Figure R.2 : RAND-Exemple de résultat

bres aléatoires. Chaque série contient trois colonnes de cinq nombres, chaque colonne étant générée par une instruction RAND différente. En d'autres termes, chaque série affiche cinq nombres aléatoires pour chacune des valeurs des instructions suivantes :

```
RAND 0
RAND 10
RAND 20
```

Le programme est contrôlé par trois boucles FOR imbriquées. La boucle externe débute à la ligne 20 et génère les deux séries. La boucle intermédiaire, ligne 40, donne trois colonnes de nombres pour chaque série. Enfin la boucle interne, ligne 70, génère cinq nombres aléatoires par colonne. L'instruction RAND se trouve à l'intérieur de la boucle intermédiaire, ligne 50 :

```
50 RAND R*10
```

La fonction RND se trouve dans la boucle interne, à la fin de l'instruction PRINT de la ligne 80.

En examinant le résultat de ce programme, on voit que les deux colonnes appelées RAND 10 sont identiques (série 1 et série 2) ; de même, les deux colonnes appelées RAND 20 contiennent des séquences de nombres identiques. Ceci prouve que l'ordinateur génère bien des séquences de nombres aléatoires identiques pour toute valeur de RAND différente de zéro.

Par contre, les deux colonnes appelées RAND 0 sont différentes. Lorsque la valeur de RAND est égale à 0, la fonction RND est initialisée, mais le point de départ de la séquence de nombres aléatoires qu'elle génère est inconnu.

Notes et commentaires

- Si l'on met l'ordinateur en mode rapide (en entrant la commande FAST), et que l'on fait tourner le programme de la Figure R.1 une seconde fois, on obtient un résultat intéressant. *Les trois* colonnes des deux séries sont identiques deux à deux, y compris les deux colonnes appelées RAND 0. Ce résultat inattendu donne un aperçu du fonctionnement de l'instruction RAND 0. Pour déterminer le point de départ impré-

visible d'une série de nombres aléatoires, l'ordinateur utilise une de ses propres *variables système*. Cette variable, appelée COLONNES, permet la tabulation du nombre de colonnes affichées sur l'écran. En mode lent, cette valeur change un certain nombre de fois entre la génération de la première et de la seconde colonne de RAND 0. Cependant, en mode rapide, l'ordinateur n'envoie des informations sur l'écran que lorsqu'il est prêt à afficher la totalité des résultats. La valeur de COLONNES ne change donc pas et génère le même point de départ pour les deux colonnes RAND 0.

REM (mot clé ; touche [E])

La commande REM (de "remarque") permet de documenter les programmes. On peut écrire après le mot clé REM n'importe quel type de commentaire ou d'information qui sera utile à la compréhension du programme. Par exemple :

```
10 REM CE PROGRAMME VOUS AIDE A EQUILIBRER
   VOTRE BUDGET
```

Les commandes REM ne produisent aucun effet ; au cours du déroulement du programme, l'ordinateur les ignore. Les lignes REM peuvent apparaître n'importe où dans un programme et pas seulement au début. Avec seulement 2 K de mémoire, on doit malheureusement faire attention au nombre et à la longueur des lignes REM dans un programme (elles occupent de l'espace mémoire même si elles n'ont aucune incidence sur son déroulement). On doit, en particulier dans les longs programmes, établir un compromis entre la clarté due à la présence des instructions REM et la nécessité de préserver l'espace mémoire.

Pour plus d'exemples sur l'utilisation de la commande REM, voir le programme de la rubrique GOSUB (Figures G.1 et G.2).

Notes et commentaires

- Une commande REM située en début de programme peut être utilisée pour délimiter l'emplacement d'un *sous-programme en code machine* (voir les rubriques PEEK, POKE et USR).

RETURN (mot clé ; touche [Y])

L'instruction RETURN marque la fin d'un sous-programme. Elle donne l'ordre à l'ordinateur de renvoyer le programme à la première ligne suivant l'instruction GOSUB qui a appelé le sous-programme. Pour avoir une description détaillée de cet exemple, voir la rubrique GOSUB.

RND (fonction ; touche [T])

Chaque appel de la fonction RND donne un nombre aléatoire, R, tel que :

```
O <= R < 1
```

Les nombres générés par la fonction RND sont en fait les résultats d'opérations complexes exécutées par l'ordinateur. Cela signifie qu'ils ne sont pas réellement aléatoires ; mais ils semblent suffisamment aléatoires pour bon nombre de programmes.

Les limites choisies pour l'ensemble des nombres donnés par la fonction RND (0 et 1), ne conviennent pas à toutes les applications qui nécessitent des nombres aléatoires. La formule suivante est couramment employée pour modifier ces limites. Elle fournit un nombre entier aléatoire, R, compris entre L (limite inférieure) et H (limite supérieure) inclus :

```
LET R= INT ((H-L+1)* RND )+L
```

Pour des entiers aléatoires, R, compris entre 1 et H inclus, la formule se réduit à :

```
LET R= INT (H* RND )+1
```

Dans cette dernière formule, l'expression

```
H* RND
```

donne un nombre compris entre 0 et H. En prenant la valeur entière de l'expression :

INT (H * **RND**)

on obtient un nombre entier compris entre 0 et H-1. Enfin, pour avoir les limites désirées 1 à H, on ajoute 1.

Exemple de programme

Le programme de la Figure R.3 montre un des algorithmes nécessaires à une partie de cartes sur ordinateur (algorithme de mélange de cartes). Le résultat de ce programme ne ressemble guère à une étape de partie de cartes car les cartes ne sont pas nommées mais seulement numérotées de 1 à 52. Ces nombres sont donc "battus" puis affichés tels quels sur l'écran. (L'exemple de la rubrique TO s'appuie sur ce programme. Avec certaines instructions supplémentaires, il affiche les cartes identifiées par leur nom.)

```
 10 PRINT " MELANGE D UN JEU DE
52 CARTES"
 20 DIM D(52)
 30 FOR I=1 TO 52
 40 LET D(I)=I
 50 NEXT I
 60 FOR I=1 TO 52
 70 LET R=INT (52*RND)+1
 80 LET H=D(I)
 90 LET D(I)=D(R)
100 LET D(R)=H
110 NEXT I
120 LET R=4
130 LET C=0
140 FOR I=1 TO 52
150 PRINT AT R,C;D(I)
160 LET R=R+1
170 IF R<=16 THEN GOTO 200
180 LET R=4
190 LET C=C+10
200 NEXT I
```

Figure R.3 : RND-Exemple de programme

Le programme de mélange de cartes stocke le paquet dans le tableau D. La ligne 20 définit ce tableau :

```
20 DIM D(52)
```

La suite du programme est constituée de trois boucles FOR successives qui exécutent les tâches suivantes :

1. créer le paquet (lignes 30 à 50)
2. battre les cartes (lignes 60 à 110)
3. afficher les cartes battues suivant leur numéro (lignes 120 à 200)

La boucle FOR intermédiaire qui met en œuvre l'algorithme de mélange montre, ligne 70, un exemple d'utilisation de la fonction RND :

```
70 LET R=INT(52*RND)+1
```

```
MELANGE D UN JEU DE 52 CARTES

47        40        23        41
46        22        27        36
35        29        32        42
2         18        38        25
34        12        49        24
21        50        3         9
31        26        1         45
10        6         5         37
17        44        39        13
4         15        11        43
28        7         16        48
8         52        20        51
33        30        19        14
```

Figure R.4 : RND-Exemple de résultat

Cette ligne prend un nombre aléatoire compris entre 1 et 52 et le stocke dans la variable R. Les trois lignes suivantes permutent les cartes entre les paquets représentés par D(I) et D(R). (Ce procédé de permutation sera plus facile à comprendre après étude du programme de *tri* de la rubrique "Algorithme".) La valeur de D(I) est d'abord stockée dans une variable "tampon", H :

```
80 LET H=D(I)
```

Ensuite on attribue à D(I) la valeur D(R) du nombre pris au hasard :

```
90 LET D(I)=D(R)
```

Enfin, on attribue à D(R) la valeur initiale de D(I) stockée dans H :

```
100 LET D(R)=H
```

Pendant que la boucle FOR incrémente la variable de contrôle I de 1 à 52, chaque carte est permutée avec une carte prise au hasard dans le paquet. Quand I atteint la valeur 52, le paquet est bien battu, comme on peut le voir Figure R.4.

RUN (mot clé ; touche [R])

Le mot clé RUN, généralement utilisé en mode immédiat, donne l'ordre à l'ordinateur de commencer à exécuter les instructions du programme en mémoire. Si on entre la commande :

```
RUN
```

l'exécution commence à la première ligne du programme. On peut d'autre part entrer la commande :

```
RUN N
```

dans laquelle N est un numéro de ligne. L'exécution commence alors à la ligne N. Dans les deux cas, l'ordinateur commence par effacer de sa mémoire les noms et les valeurs de toutes les variables laissées par les programmes précédents, puis il commence l'exécution du programme.

SAVE (mot clé ; touche [S])

La commande SAVE donne l'ordre à l'ordinateur de sauvegarder un programme sur une bande magnétique. Cette commande prend la forme suivante :

```
SAVE "NOM"
```

dans laquelle NOM peut être n'importe quel nom que l'on choisit de donner au programme (la chaîne suivant l'instruction SAVE ne peut pas être vide).

Pour transférer un programme sur une bande magnétique et le sauvegarder, appliquer la méthode suivante :

1. Utiliser les prises et les fils livrés avec l'ordinateur. Relier la prise "MIC"située sur le côté gauche de l'ordinateur à la prise "micro" du magnétophone.

2. Mettre le volume du magnétophone à environ la moitié de sa puissance et la tonalité au maximum pour les aigus et au minimum pour les basses.

3. Mettre la cassette dans le magnétophone et rembobiner si nécessaire jusqu'à l'endroit où l'on veut stocker le programme. L'enregistrement d'un programme sur cassette se fait comme un enregistrement de musique : si l'on enregistre un programme sur une portion de bande où un programme différent a été stocké antérieurement, l'ancien programme est perdu et le nouveau prend sa place. (Un magnétophone équipé d'un compteur peut être utile pour enregistrer plusieurs programmes sur la même cassette.)

4. Taper la commande SAVE avec le nom donné au programme, mais *ne pas* taper la touche ENTER tout de suite.

5. Mettre en marche le magnétophone en mode enregistrement.

6. Attendre quelques secondes puis appuyer sur la touche ENTER du clavier. L'écran devient gris-blanc et reste ainsi pendant quelques secondes. Ensuite, pendant que l'ordinateur envoie le programme sur le magnétophone, on voit apparaître sur l'écran d'épaisses lignes horizontales (environ 2,5 cm), noires et tremblotantes. Lorsque les lignes disparaissent, on voit dans le coin inférieur gauche de l'écran le message :

   ```
   0/0
   ```

   Cela signifie que l'enregistrement est terminé ; on peut arrêter le magnétophone. Pour s'assurer de la bonne marche de l'opération, on peut maintenant rembobiner la bande et charger le programme sur l'ordinateur (voir la rubrique LOAD).

Il est toujours utile de faire deux copies de tous les programmes que l'on veut garder et de les faire sur deux cassettes différentes. Mettre la copie de sauvegarde en lieu sûr et garder l'autre pour travailler. Si un problème se pose lors d'une manipulation, utiliser la copie de sauvegarde uniquement pour créer une nouvelle copie de travail.

Notes et commentaires

La plupart des ordinateurs ainsi que la plupart des versions du BASIC possèdent des mots clés qui permettent, en plus du stockage de programmes, le stockage de *données* sur cassette magnétique ou sur disque souple. Certains BASIC par exemple ont des instructions qui permettent de stocker directement des données dans le programme ; on peut ensuite inclure dans ce même programme des instructions qui lisent ces données, chaque fois que c'est nécessaire, au cours de son déroulement. D'autres BASIC ont la possibilité de

créer des fichiers externes permanents, ces fichiers contiennent des données mais pas de lignes de programme. On peut ensuite écrire un programme qui "ouvre"ce genre de fichier et introduit son contenu dans la mémoire de l'ordinateur pour l'utiliser en cours d'exécution.

Bien que le ZX 81 ne possède pas de commandes spécifiques pour la sauvegarde de données, il existe une méthode pour les stocker sur cassette et les réutiliser. Cette méthode est un peu délicate mais vaut la peine d'être apprise quand on projette d'écrire des programmes qui traitent une quantité de données assez importante. La commande SAVE ne stocke pas seulement un programme sur la bande magnétique ; elle stocke également, au moment où l'on sauvegarde le programme, toutes les variables qui sont dans la mémoire de l'ordinateur ainsi que leurs valeurs. Cela signifie qu'au moment où le programme est à nouveau chargé, on charge également les variables et leurs valeurs.

L'astuce consiste à trouver un moyen commode pour *utiliser* ces données. On sait que la commande RUN efface de la mémoire de l'ordinateur les variables et leurs valeurs avant exécution du programme. Il ne faut donc pas taper la touche RUN si on veut sauvegarder les données. On peut cependant lancer le programme avec une commande GOTO immédiate.

Pour comprendre exactement le fonctionnement de cette méthode, il faut regarder l'exemple de programme décrit dans la rubrique DIM (Figure D.1). Ce programme, destiné à un professeur qui veut enregistrer et comparer les résultats de ses élèves, comprend deux séries de données : les noms des élèves stockés dans le tableau N$, et les résultats de leurs épreuves stockés dans le tableau S. Le programme génère un histogramme à partir de ces données (voir l'exemple de la Figure D.2). Les étapes suivantes exposent la procédure à suivre pour stocker et réutiliser les données de ce programme :

1. Entrer le programme dans l'ordinateur et le faire tourner. Faire un essai en entrant les noms et les résultats des élèves de la Figure D.2. Il y a 20 élèves sur la liste, on entre donc la valeur 20 en réponse à la première question :

   ```
   NOMBRE D'ELEVES?
   ```

   Ensuite le programme réclame l'entrée du nom et des résul-

tats de chaque élève un par un jusqu'à l'entrée complète de la liste. L'ordinateur dessine l'histogramme sur l'écran.

2. Ensuite, sauvegarder le programme sur bande magnétique pendant que les données obtenues sont encore en mémoire. On peut par exemple appeler ce programme "TEST NO.1" :

   SAVE "TEST NO.1"

   Il ne faut pas oublier de sauvegarder non seulement le programme mais également toutes les variables obtenues après son exécution, ainsi que leurs valeurs.

3. On imagine maintenant que plusieurs semaines ont passé et qu'on a utilisé l'ordinateur pendant cette période pour d'autres travaux. (Débrancher puis rebrancher l'ordinateur, ou entrer la commande NEW, pour avoir l'assurance qu'il ne reste en mémoire aucune trace du programme ou de ses données.) Charger le programme de la cassette sur l'ordinateur :

   LOAD "TEST NO.1"

   Il faut faire attention à ne pas utiliser la commande RUN car cela entraînerait la perte des données chargées avec le programme. Il faut sauter la partie "entrée de données"(-dans la mesure où l'on ne veut pas entrer de nouvelles données) et envoyer le programme juste au début du bloc d'instructions qui génère l'histogramme. Ces instructions commençant ligne 120, on entre donc la commande :

   GOTO 120

   Si tout se passe bien, on voit l'histogramme de la Figure D.2 réapparaître sur l'écran.

4. Pour finir, on imagine que l'on veuille utiliser le programme pour créer un nouvel histogramme relatif aux mêmes élèves mais avec des résultats différents (TEST NO.2). On veut éviter de retaper les noms des élèves. Ceci est possible en modi-

fiant légèrement le programme ; il faut éliminer toutes les lignes qui donnent de nouvelles valeurs au tableau de noms, N$. On commence par effacer les lignes 10 à 40 puisque les instructions DIM ne sont plus d'aucune utilité. (Pour effacer une ligne de programme, il suffit de taper le numéro de la ligne et la touche ENTER.) Ensuite on élimine les lignes 60 et 70, lignes d'entrée des noms à l'intérieur de la boucle FOR (ligne 50). Il faut aussi modifier la ligne 55 pour que le programme affiche le nom de chaque élève et ne demande que sa note :

55 **PRINT AT** 20,0;N$(I)

On peut aussi inclure dans le programme quelques lignes REM pour rappeler qu'il ne faut pas entrer la commande RUN. La Figure S.1 montre le programme entièrement modifié.

Bien que le programme ait été modifié, les données (noms et notes

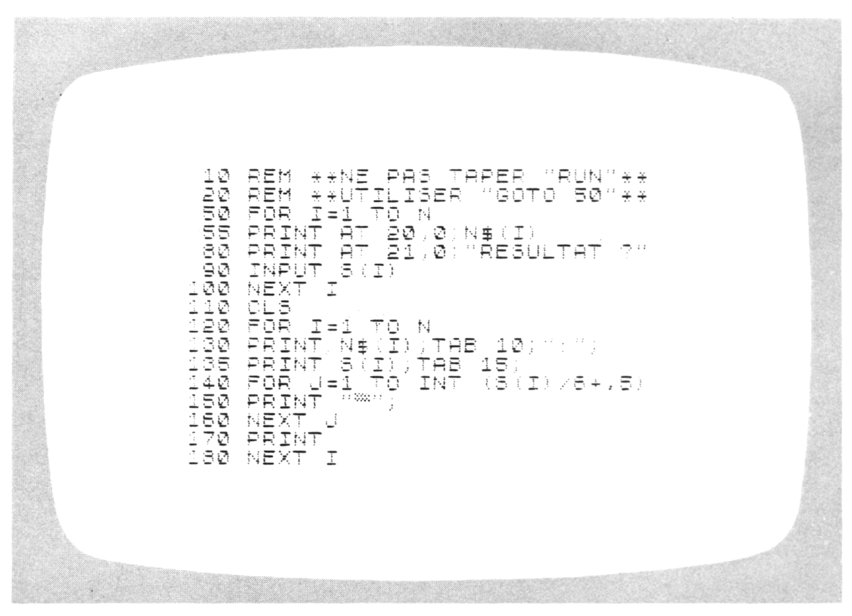


Figure S.1 : SAVE-Exemple de programme

des élèves) doivent être intactes dans la mémoire de l'ordinateur. Si l'on commence l'exécution du programme en entrant la commande :

GOTO 50

on obtient une série de 20 demandes d'entrée de nouveaux résultats. A chaque demande, l'ordinateur affiche le nom d'un élève ; par exemple :

PASCAL
NOTE ?

Il suffit de taper le nouveau résultat de l'élève. Lorsque les 20 résultats sont entrés, l'ordinateur affiche un nouvel histogramme qui représente les nouveaux résultats de ce même groupe d'élèves (voir exemple de la Figure S.2).

Si l'on veut maintenant sauvegarder ce nouvel ensemble de résultats, on peut le faire en sauvegardant la version modifiée du programme, que l'on appelle "TEST NO.2". On peut également

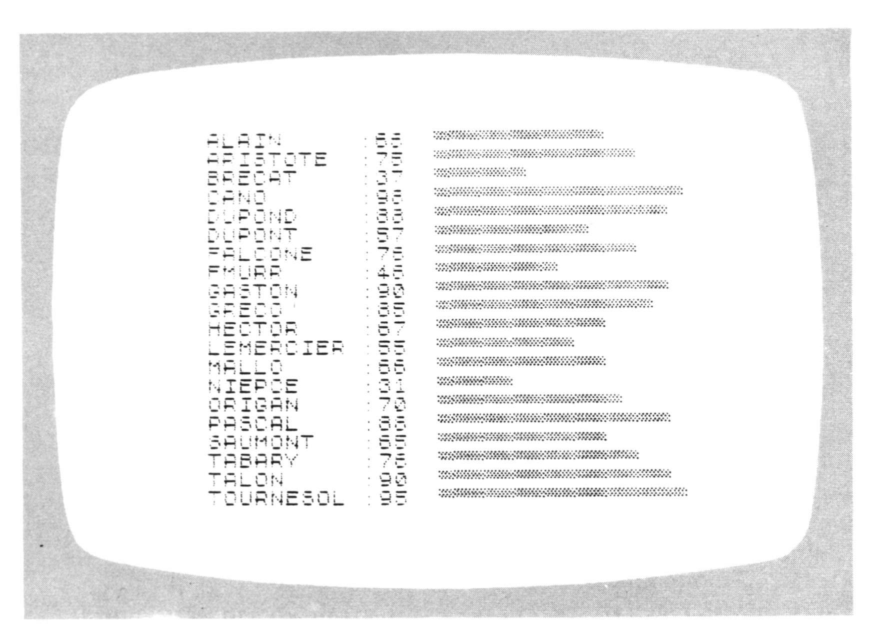


Figure S.2 : SAVE-Exemple de résultat

utiliser cette version modifiée pour entrer des résultats ultérieurs relatifs au même groupe d'élèves.

SCROLL (mot clé ; touche [B])

L'instruction SCROLL déplace d'une ligne vers le haut les lignes affichées sur l'écran. La ligne supérieure disparaît (définitivement) et une ligne vide apparaît en bas de l'écran. Chaque instruction PRINT suivante place l'information sur cette ligne vide.

Exemple de programme

Le programme de la Figure S.3 est une variante de l'exemple de programme de la rubrique CONT (voir les Figures C.10 à C.12). Ce

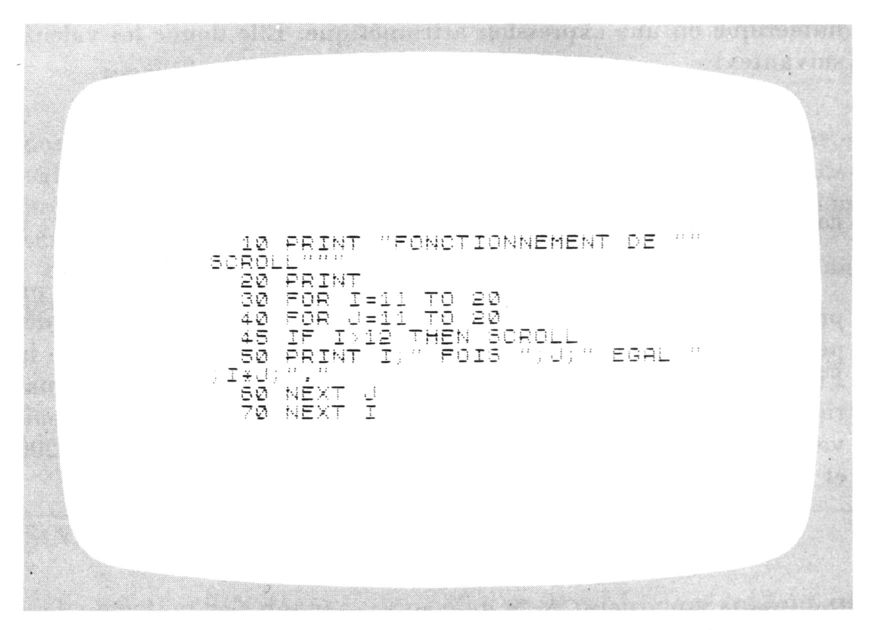


Figure S.3 : SCROLL-Exemple de programme

programme affiche les résultats de multiplications, de "11 fois 11"à "20 fois 20". Dans cette variante, l'ordinateur exécute l'instruction SCROLL dès que l'écran est plein. L'instruction SCROLL est contrôlée par la ligne 45 :

```
45 IF I>12 THEN SCROLL
```

Faire tourner le programme pour voir le fonctionnement de l'instruction SCROLL.

SGN (fonction ; touche [F])

La fonction SGN identifie le signe d'un nombre quelconque. Elle prend la forme :

```
SGN N
```

dans laquelle N est une valeur numérique littérale, une variable numérique ou une expression arithmétique. Elle donne les valeurs suivantes :

-1 si $N < 0$
0 si $N = 0$
+1 si $N > 0$

Exemple de programme

La fonction SGN peut être utile quand il faut aiguiller un programme vers une tâche définie. Le choix de cet "aiguillage"dépend du signe de l'argument de la fonction. Le programme de la Figure S.4 illustre l'emploi de la fonction SGN dans ce type de situation. Trois sous-programmes sont réservés pour les situations suivantes : $N<0$, $N=0$ et $N>0$ respectivement situés lignes 100, 200 et 300. La ligne 50 appelle le sous-programme :

```
50 GOSUB (SGN N+2)*100
```

Puisque la fonction SGN N donne un nombre entier compris entre -1 et +1, l'expression :

```
10 REM ** SOUS PROGRAMMES LIES
20 REM ** AU SIGNE DE N.
30 REM
40 INPUT N
50 GOSUB (SGN N + 2)*100
60 STOP
100 PRINT "N EST NEGATIF"
110 RETURN
200 PRINT "N EST NUL"
210 RETURN
300 PRINT "N EST POSITIF"
310 RETURN
```

Figure S.4 : SGN-Exemple de programme

SGN N+2

donne un nombre entier compris entre 1 et 3. Si l'on multiplie cette valeur par 100, on obtient le point de départ de l'un des trois sous-programmes. (Voir la rubrique GOSUB pour avoir une explication sur les appels calculés de sous-programmes.)

Si l'on n'utilisait pas la fonction SGN, il faudrait trois lignes au programme pour choisir le sous-programme à appeler :

```
45 IF N < 0 THEN GOSUB 100
50 IF N = 0 THEN GOSUB 200
55 IF N > 0 THEN GOSUB 300
```

SIN (fonction ; touche [Q])

La fonction SIN donne le sinus d'un angle quelconque (négatif ou positif) exprimé en radians.

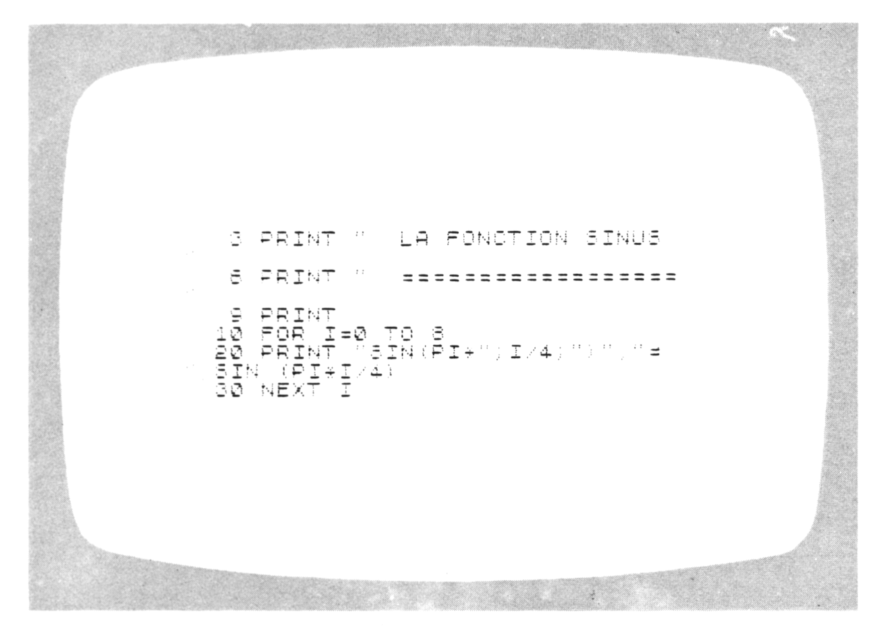


Figure S.5 : SIN-Exemple de programme

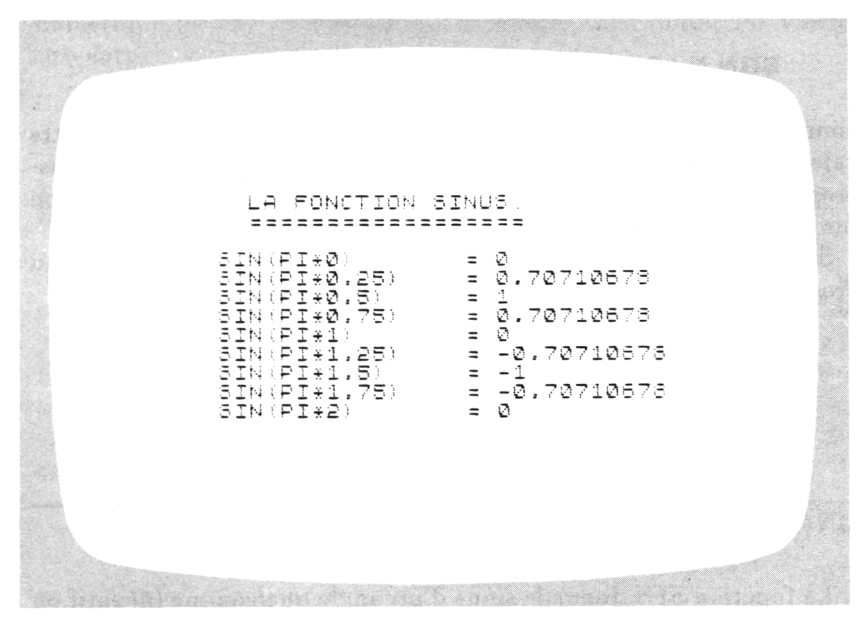


Figure S.6 : SIN-Exemple de résultat

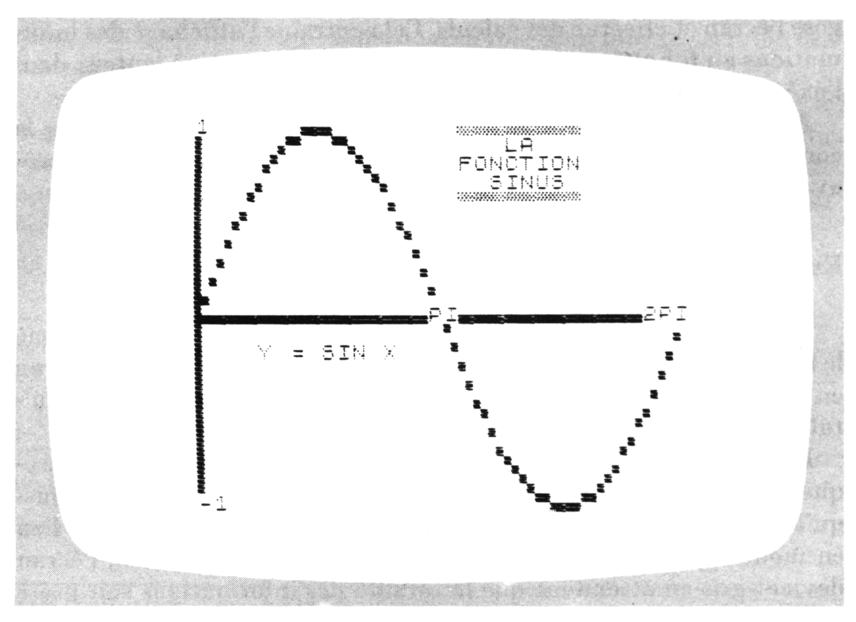


Figure S.7 : SIN-Tracé de courbe

Exemple de programme

Le programme de la Figure S.5 affiche une série de sinus dont les arguments varient de 0 à 2 π. On voit le résultat de ce programme Figure S.6.

Notes et commentaires

- La Figure S.7 montre la courbe de la fonction sinus pour x variant de 0 à 2 π.
- Pour plus d'information sur les fonctions trigonométriques, voir les rubriques COS et TAN.

SLOW (shift ; touche [D])

La commande SLOW met l'ordinateur en mode lent. Quand il est dans ce mode, l'ordinateur exécute simultanément deux tâches : il

gère l'écran et effectue des calculs. Cela entraîne l'affichage des informations au fur et à mesure des résultats et une certaine lenteur dans l'exécution des tâches.

A l'initialisation, l'ordinateur est en mode lent. On utilise donc la commande SLOW uniquement pour revenir en mode lent après avoir utilisé la commande FAST (voir la rubrique FAST).

Exemple de programme

Le programme de la Figure S.8 met en évidence ce qui différencie le mode lent du mode rapide. Ce programme occulte deux fois l'écran en le remplissant de caractères blancs en vidéo inverse. Cette occultation est réalisée par le sous-programme de la ligne 200.

La première occultation a lieu en mode lent (ligne 30). En conséquence, l'ordinateur envoie un par un les caractères sur l'écran jusqu'à ce qu'il soit complètement noir. La deuxième occultation a lieu en mode rapide. Dès que la commande FAST est exécutée, l'écran devient gris en attendant que la totalité des informations soit prête

```
 10 PRINT TAB 9;"MODE LENT "
 20 PRINT
 30 SLOW
 40 GOSUB 200
 50 PRINT AT 21,0;"CONTINUER ?"
 60 INPUT A$
 70 CLS
 80 PRINT TAB 9;"MODE RAPIDE"
 90 PAUSE 100
100 FAST
110 GOSUB 200
120 STOP
200 FOR I=1 TO 640
210 PRINT "█";
220 NEXT I
230 RETURN
```

Figure S.8 : SLOW-Exemple de programme

à être affichée. Ensuite, l'ordinateur envoie d'un bloc l'ensemble des informations à afficher et l'on voit l'écran s'occulter.

On voit ce qui différencie les deux modes en faisant tourner le programme. En résumé, quels que soient les calculs que l'ordinateur doit exécuter, les différences entre le mode lent et le mode rapide sont : la vitesse d'exécution et la synchronisation (immédiate ou différée) de l'affichage sur l'écran.

Sous-programme (vocabulaire informatique)

Un sous-programme est un ensemble d'instructions, séparé du corps du programme principal, et destiné à être exécuté par l'intermédiaire d'une commande GOSUB (voir la rubrique GOSUB).

SQR (fonction ; touche [H])

```
 10 PRINT TAB 4;"LE THEOREME DE
PYTHAGORE"
 11 PRINT
 12 PRINT
 13 PRINT TAB 9;" 2    2    2"
 14 PRINT TAB 9;"A  + B  = C"
 20 LET A$="COTE A: "
 30 LET B$="COTE B: "
 40 PRINT AT 21,0;A$
 50 INPUT A
 60 PRINT AT 21,0;B$
 70 INPUT B
 80 LET C=SQR (A*A+B*B)
 90 PRINT AT 6,10;A$;A
100 PRINT AT 7,10;B$;B
110 PRINT AT 9,8;"HYPOTENUSE =
";C
120 PRINT AT 21,0;"CONTINUER ?"
130 INPUT R$
140 CLS
150 GOTO 10
```

Figure S.9 : SQR-Exemple de programme

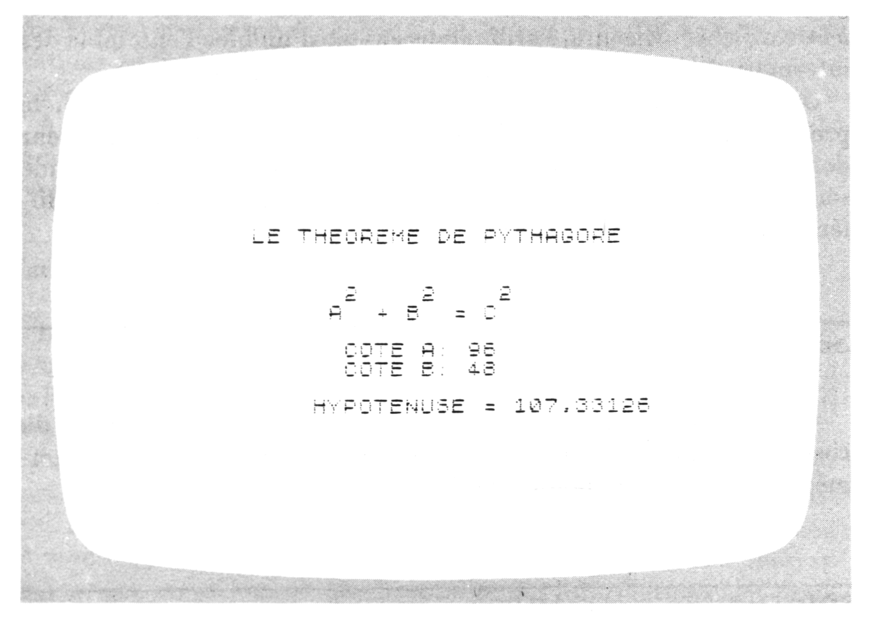


Figure S.10 : SQR-Exemple de résultat

La fonction SQR donne la racine carrée d'un nombre supérieur ou égal à zéro.

Exemple de programme

La Figure S.9 montre un programme qui permet de calculer la valeur de l'hypoténuse d'un triangle rectangle en connaissant la valeur de ses deux autres côtés, A et B. La ligne 80 calcule l'hypoténuse, C, en utilisant la fonction SQR :

```
80 LET C= SQR (A*A+B*B)
```

Un exemple de résultat illustre ce programme Figure S.10.

STEP (shift ; touche [E])

Le mot clé STEP, élément de l'instruction FOR, indique de quelle valeur la variable de contrôle doit être incrémentée (ou décrémentée)

à chaque itération de la boucle. Le mot clé STEP est facultatif dans une instruction FOR ; dans ce cas, l'incrémentation *par défaut* est égale à 1.

Par exemple, l'instruction FOR suivante introduit une boucle dont la variable de contrôle est I :

```
FOR I=0 TO 25
```

Dans ce cas, la boucle est réitérée 26 fois ; I prend les valeurs suivantes 0,1,2,3, ..., 25. Si l'on ajoute le mot clé STEP à cette instruction, on change à la fois la série de valeurs que prend I et le nombre d'itérations de la boucle :

```
FOR I=0 TO 25 STEP 5
```

La boucle est seulement répétée 6 fois et I prend les valeurs ; 0, 5, 10, 15, 20 et 25.

Le mot clé STEP peut aussi déterminer des valeurs d'incrémentation fractionnaires ou négatives. Par exemple, dans la boucle qui suit, la variable de contrôle I varie de -1 à +1 par pas de 0,1 :

```
FOR I=-1 TO 1 STEP .1
```

I prend les valeurs -1 ; -0,9 ; -0,8 ; ... ; 0 ; 0,1 ; 0,2 ; pour finir, l'instruction FOR suivante définit une variable de contrôle de *décrémentation* :

```
FOR I=20 TO 0 STEP -2
```

On remarque que le nombre placé devant TO est supérieur au nombre placé après ; en l'absence du mot clé STEP, la boucle FOR n'aurait eu aucun effet. Cependant, le mot STEP définit une valeur d'incrémentation égale à -2. La variable de contrôle prend donc des valeurs *décroissantes* comprises entre 20 et 0, c'est-à-dire 20, 18, 16, 14, ..., 0.

Exemple de programme

La Figure S.11 propose un programme amusant dans lequel apparaît un cafard qui illustre l'utilisation du mot clé STEP. Ceci est un

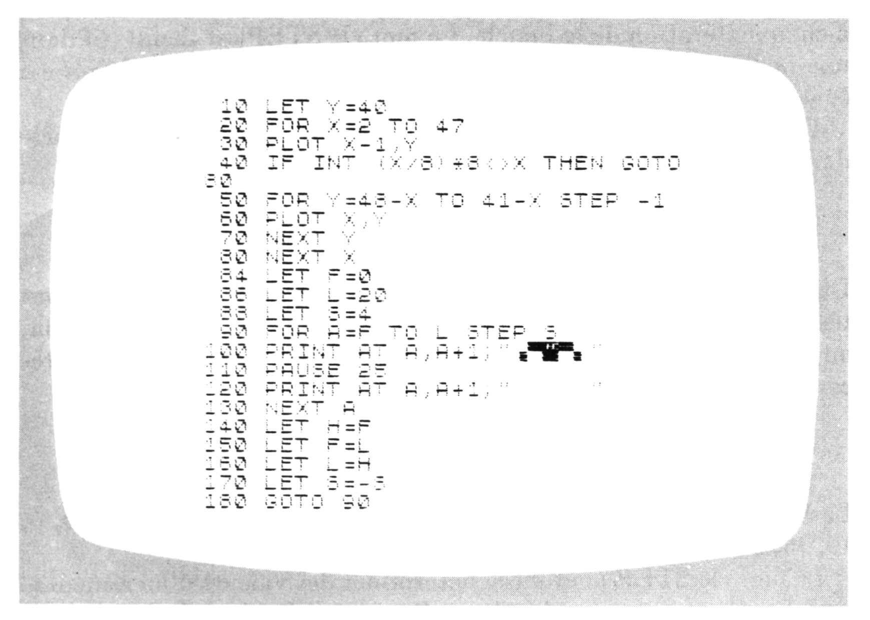


Figure S.11 : STEP-Exemple de programme

exemple simple d'animation d'écran. En faisant tourner le programme, on voit tout d'abord apparaître un escalier. Puis un "gentil petit cafard" commence à monter et descendre les escaliers (voir Figure S.12).

Dans ce programme, deux boucles imbriquées (lignes 20 à 80) ont pour tâche de dessiner les escaliers. La boucle FOR des lignes 90 à 130 génère le cafard et son déplacement. Il faut bien comprendre cette dernière boucle qui illustre deux aspects de l'utilisation du mot clé STEP.

L'instruction FOR introduit la variable de contrôle A :

90 **FOR** A = F **TO** L **STEP** S

Cette variable détermine à l'intérieur de la boucle les coordonnées d'adresse de deux instructions PRINT AT. Ces instructions (lignes 100 et 120) placent le cafard sur une des marches de l'escalier puis l'effacent. Le cafard est réalisé à l'aide de cinq caractères graphiques : SHIFT-3, SHIFT-7, SHIFT-P, SHIFT-7, SHIFT-3. Les yeux (SHIFT-P) sont des guillemets en vidéo inverse.

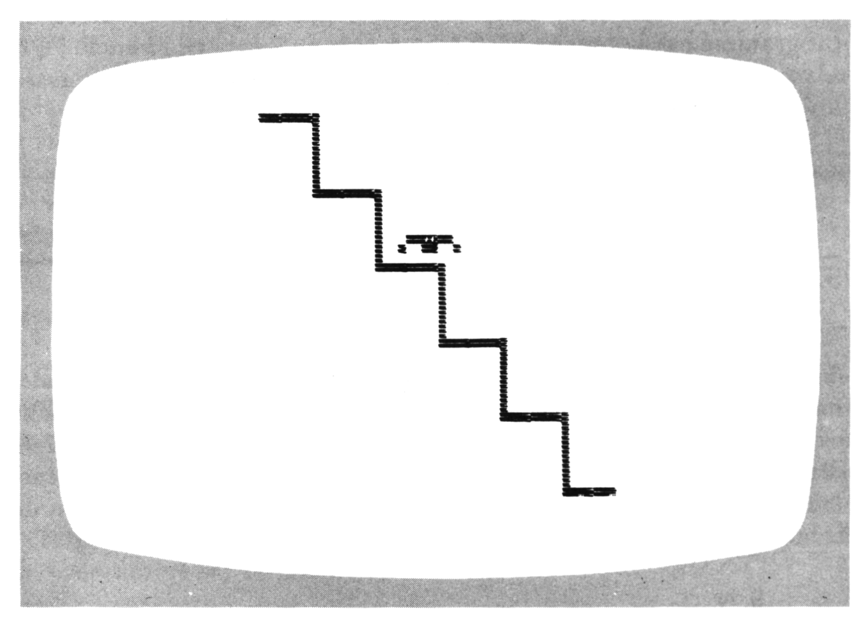


Figure S.12 : STEP-Exemple de résultat

On remarque que toutes les valeurs de l'instruction FOR sont représentées par des variables. Ces deux variables (F pour "premier" et L pour "dernier") ainsi que l'argument de STEP sont initialisés par les lignes 84 à 88. Ces valeurs initiales donnent à la boucle FOR la forme suivante :

```
90 FOR A=0 TO 20 STEP 4
```

A ce stade du programme, la variable de contrôle A prend les valeurs 0, 4, 8, ..., 20 qui permettent au cafard de descendre l'escalier.

Après exécution complète de la boucle FOR (quand le cafard a atteint le bas de l'escalier), les lignes 140 à 160 permutent les valeurs de F et de L, et la ligne 170 inverse le signe de S. L'instruction GOTO, ligne 180, renvoie ensuite le programme au début de la boucle. Avec les nouvelles valeurs de F, L et S, la boucle FOR prend la forme :

```
90 FOR A=20 TO 0 STEP -4
```

et A prend les valeurs 20, 16, 12, ..., 0 qui permettent au cafard de

remonter les escaliers. Le processus continue jusqu'à l'arrêt du programme par la touche BREAK. A chaque fois que la boucle FOR a accompli sa tâche, les instructions des lignes 140 à 170 inversent les valeurs des variables et la boucle repart.

STOP (shift ; touche [A])

L'instruction STOP commande à l'ordinateur l'arrêt du déroulement d'un programme. On peut inclure une instruction STOP dans un programme ou l'entrer à partir du clavier en commande immédiate au cours de son déroulement. Si l'ordinateur rencontre une instruction STOP au cours d'un programme, il termine l'exécution de celui-ci avec le message :

```
9/N
```

dans lequel N est la ligne que l'ordinateur exécutait quand le programme a rencontré l'instruction STOP. Si l'on arrête le programme en entrant l'instruction STOP, un message d'erreur différent apparaît :

```
D/N
```

N est encore la ligne que l'ordinateur exécutait quand le programme s'est arrêté.

La touche BREAK et la commande STOP peuvent être toutes deux utilisées pour mettre fin à l'exécution d'un programme, mais elles sont utilisées à des moments différents. La touche BREAK arrête un programme au cours de n'importe quelle activité sauf pendant l'entrée de données. Par contre, quand l'ordinateur attend une entrée de données (c'est-à-dire quand il exécute une instruction INPUT), on doit utiliser la commande STOP pour arrêter le programme. Si l'ordinateur attend une entrée de données numériques, on voit apparaître le repère L dans le coin inférieur gauche de l'écran ; il suffit pour arrêter le programme d'entrer la commande STOP (SHIFT-A et ENTER). Si l'ordinateur attend l'entrée de

chaînes de caractères, le processus est légèrement plus compliqué. On voit de nouveau apparaître le repère L, mais cette fois-ci, entre guillemets. Pour entrer la commande STOP, on doit d'abord utiliser les touches flèche-à-gauche ou flèche-à-droite (SHIFT-5 ou SHIFT-8) pour sortir le repère-L des guillemets. On peut ensuite entrer la commande STOP pour mettre fin au programme. (Si l'on ne déplace pas le repère L, l'ordinateur reçoit la commande STOP comme un caractère d'entrée et continue l'exécution du programme.)

Exemple de programme

La Figure S.13 illustre une situation dans laquelle la commande STOP est nécessaire au déroulement correct du programme. Normalement, quand l'ordinateur exécute un programme ne contenant pas de commande STOP, il se contente d'exécuter chaque ligne à partir du début et jusqu'à la fin du programme. Dans ce cas, la commande STOP n'est pas nécessaire, l'ordinateur s'arrête de lui-même quand le programme est terminé.

```
  10 REM ++ PROGRAMME PRINCIPAL +
+
  20 GOSUB 100
  30 GOSUB 200
  40 GOSUB 300
  50 STOP
 100 REM ++ SOUS PROGRAMME 1
 110 RETURN
 200 REM ++ SOUS PROGRAMME 2
 210 RETURN
 300 REM ++ SOUS PROGRAMME 3
 310 RETURN
```

Figure S.13 : STOP-Exemple de programme

Cependant, la situation est différente pour un programme contenant des sous-programmes. Les sous-programmes se trouvent généralement en fin de programme; le début, parfois appelé "programme principal" appelle les sous-programmes aux moments appropriés.

La Figure S.13 montre un exemple de ce type de structure. Les lignes 10 à 50 forment le programme principal, et les sous-programmes se trouvent aux lignes 100, 200 et 300. Après l'appel de tous les sous-programmes, on doit indiquer clairement à l'ordinateur qu'il doit s'arrêter. Dans le cas contraire, le programme continue son déroulement dans la zone réservée aux sous-programmes.

Si l'on fait tourner ce programme (qui n'a pas d'action réelle), on voit à la fin du déroulement le message suivant :

```
9/50
```

Ce message indique que l'ordinateur s'est arrêté à la ligne 50. On peut voir les conséquences de l'absence de commande STOP en essayant de supprimer la ligne 50 et en faisant tourner le programme à nouveau. On obtient le message d'erreur suivant :

```
7/110
```

Le code d'erreur "7" signifie que l'ordinateur a rencontré une instruction RETURN non précédée d'une commande GOSUB. Il indique que le programme s'est déplacé par erreur dans la zone des sous-programmes.

STR$ (fonction ; touche [Y])

La fonction STR$ donne la chaîne de caractères correspondant à un argument numérique. La chaîne de caractères donnée par cette fonction est composée des caractères que l'ordinateur utiliserait pour afficher le nombre sur l'écran.

On prend par exemple les lignes suivantes :

```
LET N=157.321
LET A$= STR$ N
```

La seconde ligne stocke la valeur de chaîne "157.321" dans la variable de chaîne A$. La valeur de N et la valeur de A$ se différencient par la manière dont l'ordinateur stocke les valeurs numériques et les valeurs de chaînes. Les valeurs numériques, comme par exemple la valeur de N, sont stockées dans un système appelé *virgule flottante*. Ce système, qui stocke séparément les chiffres significatifs du nombre ("mantisse") et son exposant, est conçu pour donner un maximum de précision et de facilité dans la manipulation d'un large éventail de nombres. (Le ZX 81 peut stocker des nombres compris entre 10^{-39} et 10^{+38} avec une précision de 9 ou 10 chiffres.) D'autre part, les valeurs de chaîne telles que la valeur de A$ sont stockées caractère par caractère dans leurs équivalents code-caractère du ZX 81. Pour une explication détaillée de ce code, voir les rubriques CHR$ et CODE.

Exemple de programme

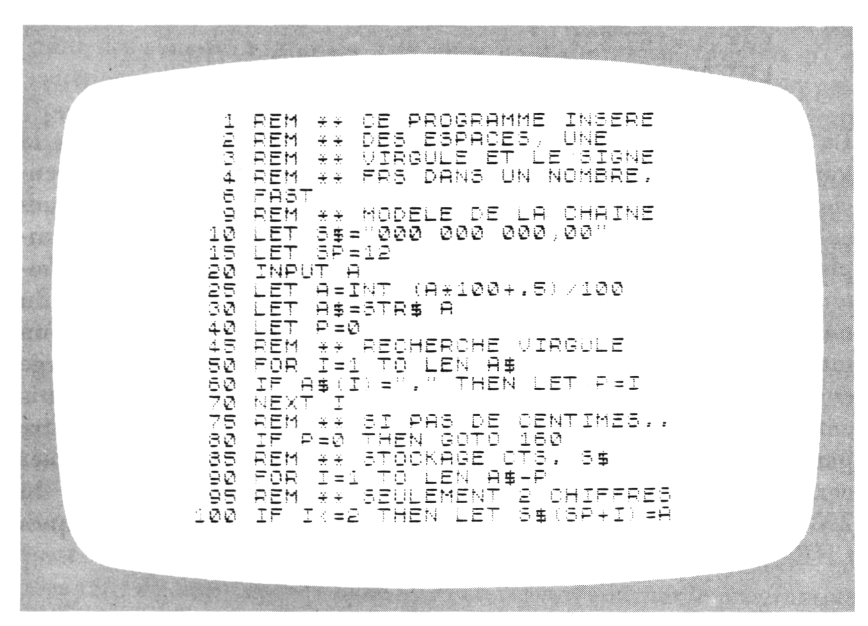
Le programme des Figures S.14 et S.15 convertit puis affiche une valeur numérique représentant des francs et des centimes en une chaîne numérique qui comprend les signes Frs, des espaces et une virgule décimale. La Figure S.16 montre une colonne de nombres générés par ce programme. Certains BASIC possèdent la commande PRINT USING qui exécute la même tâche, mais pas le BASIC du ZX 81.

Le programme est long et compliqué, essentiellement parce qu'il permet l'entrée d'une grande variété de valeurs. Il contient des commentaires REM qui aident à sa compréhension.

On commence par étudier les lignes 20 à 30 :

```
20 INPUT A
25 LET A= INT (A*100+0.5)/100
30 LET A$= STR$ A
```

La ligne 20 lit une valeur numérique à partir du clavier et la stocke dans la variable A ("somme"). La ligne 25 utilise la fonction INT pour arrondir la valeur de A à deux décimales (ceci permet d'entrer



```
  1 REM ** CE PROGRAMME INSERE
  2 REM ** DES ESPACES, UNE
  3 REM ** VIRGULE ET LE SIGNE
  4 REM ** FRS DANS UN NOMBRE.
  6 FAST
  9 REM ** MODELE DE LA CHAINE
 10 LET S$="000 000 000,00"
 15 LET SP=12
 20 INPUT A
 25 LET A=INT (A*100+.5)/100
 30 LET A$=STR$ A
 40 LET P=0
 45 REM ** RECHERCHE VIRGULE
 50 FOR I=1 TO LEN A$
 60 IF A$(I)="," THEN LET P=I
 70 NEXT I
 75 REM ** SI PAS DE CENTIMES..
 80 IF P=0 THEN GOTO 160
 85 REM ** STOCKAGE CTS. S$
 90 FOR I=1 TO LEN A$-P
 95 REM ** SEULEMENT 2 CHIFFRES
100 IF I<=2 THEN LET S$(SP+I)=A
```

Figure S.14 : STR$-Exemple de programme

```
110 NEXT I
120 LET N$=A$(1 TO P-1)
130 IF N$<>"" THEN GOTO 180
135 REM ** PAS DE FRANCS
140 LET SP=SP-1
150 GOTO 230
160 LET N$=A$
165 REM ** STOCKER FRS S$
180 FOR I=LEN N$ TO 1 STEP -1
190 LET SP=SP-1
195 REM ** METTRE ESPACES S$
200 IF INT (SP/4)*4=SP THEN LET
SP=SP-1
210 LET S$(SP)=N$(I)
220 NEXT I
225 REM ** P$=RESULTAT
227 REM ** A IMPRIMER
230 LET P$=S$(SP TO LEN S$)
240 PRINT TAB (1+LEN S$-LEN P$)
P$;" FRS"
250 GOTO 10
```

Figure S.15 : STR$-Exemple de programme, (suite)

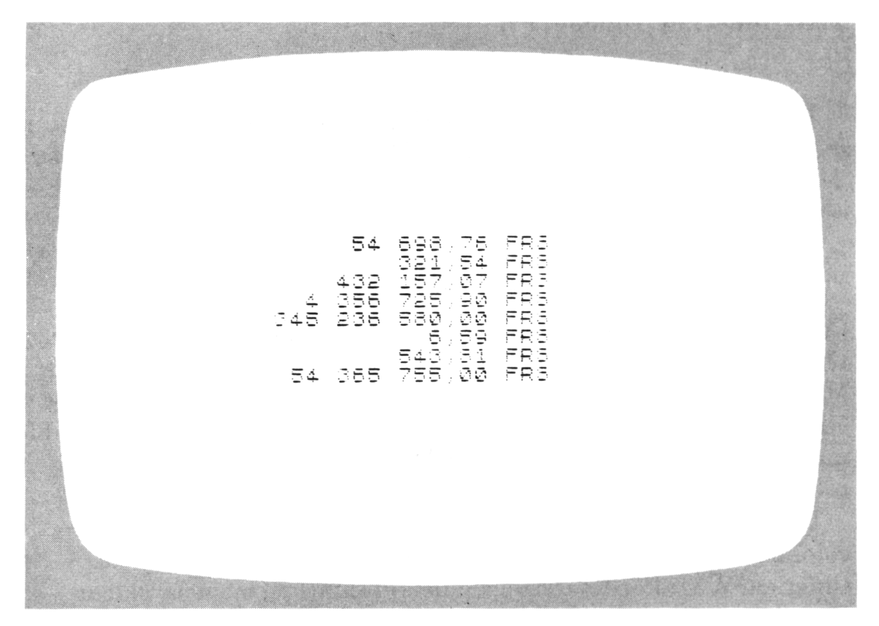


Figure S.16 : STR$-Exemple de résultat

une valeur telle que 345,987 qui sera arrondie à 345,99). La ligne 30 assigne à la variable A$ la valeur de chaîne de A donnée par la fonction STR$. La méthode du programme consiste à ajuster les caractères de A$ à la chaîne "gabarit", S$. Pour accomplir cela, le programme utilise la méthode du *découpage de chaîne* du ZX 81 (pour une description de cette méthode, voir la rubrique TO). On remarque que la ligne 10 assigne à S$ la valeur suivante :

```
10 LET S$="000 000 000,00"
```

Le programme peut manipuler des sommes pouvant atteindre neuf chiffres, mais la précision sera défectueuse pour des nombres très grands.

TAB (fonction ; touche [P])

L'instruction TAB est une fonction de programmation exclusivement utilisée avec la commande PRINT. Elle détermine le numéro de la colonne dans laquelle commence l'affichage d'un élément. Elle est suivie d'un seul argument numérique et prend la forme :

```
PRINT TAB N;"INFORMATION"
```

Cette instruction indique à l'ordinateur de placer le premier caractère d'un élément à afficher (dans le cas présent, la chaîne "INFORMATION") dans la colonne N de la ligne en cours. La valeur N peut être exprimée comme une valeur numérique littérale, une variable numérique ou une expression arithmétique. L'instruction TAB doit toujours être séparée de l'élément à afficher par un point-virgule.

L'argument N de la fonction TAB correspond à un numéro de colonne compris entre 0 et 31 (largeur totale de l'écran). Si N est supérieur à 31, l'ordinateur calcule automatiquement la valeur

N modulo 32

pour définir un numéro de colonne compris dans les limites autorisées, 0 à 31. L'expression *N modulo 32* signifie *reste* de la division de N par 32. Un argument négatif de la fonction TAB entraîne un arrêt du programme avec un message d'erreur "B".

Exemple de programme

Le programme de la Figure T.1 est un exercice destiné à illustrer l'utilisation de la fonction TAB. La Figure T.2 montre le résultat de ce programme. Celui-ci imprime sur l'écran 20 lignes qui marquent chacune l'emplacement d'un taquet de tabulation. Le message situé sur chaque ligne affiche l'argument de la fonction. Cet argument détermine la position du premier caractère du message, "T".

Les six premières lignes de la Figure T.2 montrent des taquets de tabulation correspondant à des arguments inférieurs à 32. Les lignes suivantes illustrent des manipulations d'arguments supérieurs à 32. Prenons le premier de ces exemples, TAB 35. 35 modulo 32 étant égal à 3, TAB 35 doit être équivalent à TAB 3, ce que confirme la Figure T.2.

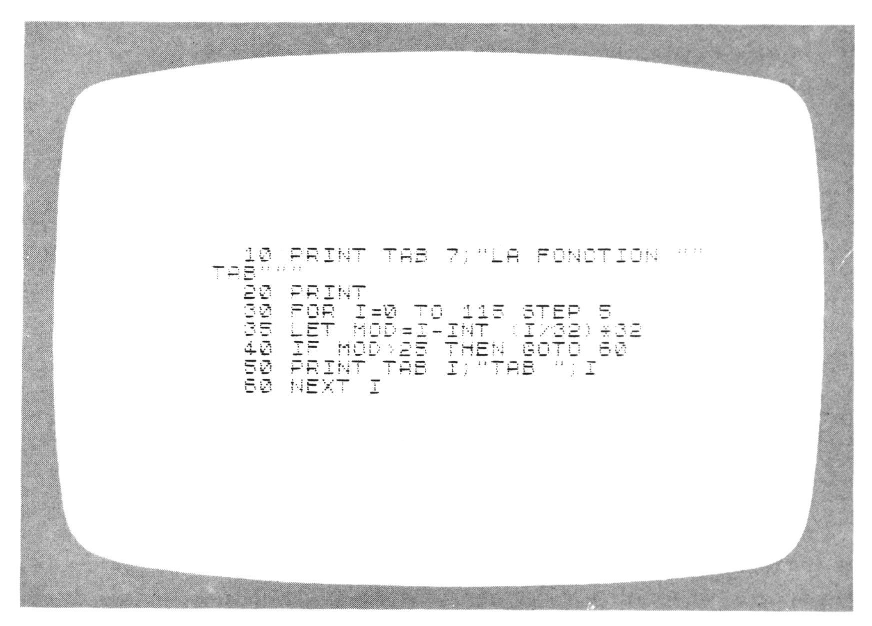


Figure T.1 : TAB-Exemple de programme

```
       LA FONCTION "TAB"

TAB 0
     TAB 5
          TAB 10
               TAB 15
                    TAB 20
                         TAB 25
   TAB 35
        TAB 40
             TAB 45
                  TAB 50
                       TAB 55
 TAB 65
      TAB 70
           TAB 75
                TAB 80
                     TAB 85
    TAB 100
         TAB 105
              TAB 110
                   TAB 115
```

Figure T.2 : TAB-Exemple de résultat

On voit dans le programme que les messages TAB sont générés par une boucle FOR qui incrémente la variable de contrôle I, de 0 à 115, par pas de 5. La ligne 50 contient l'instruction suivante :

```
50 PRINT TAB I;"TAB ";I
```

Cette instruction provoque d'abord une tabulation jusqu'à la colonne I de la ligne en cours, puis affiche deux éléments sur l'écran : la chaîne "TAB" et la valeur de I.

Tableau (vocabulaire informatique)

Un tableau est une organisation de données établie pour permettre le stockage de listes ou de tables de données. Le nom, le genre, la longueur et le nombre de dimensions du tableau sont définis dans l'instruction DIM. Chaque donnée stockée dans le tableau est affectée d'un ou de plusieurs indices qui permettent de la localiser dans le tableau. (Voir la rubrique DIM).

TAN (fonction ; touche [E])

La fonction TAN donne la tangente de n'importe quel angle exprimé en radians. La fonction tangente tend vers plus ou moins l'infini quand son argument tend vers un multiple impair de $\pm \pi/2$. Pour ces valeurs, l'ordinateur donne un message d'erreur "6 "indiquant que le résultat est un nombre trop grand pour être stocké en mémoire.

Exemple de programme

Le programme de la Figure T.3 affiche les valeurs d'une série de tangentes dont les arguments varient de 0 à 2π. La variable de con-

```
   3 PRINT TAB 6;"LA FONCTION TA
NGENTE "
   6 PRINT TAB 6;"==============
======"
   9 PRINT
  10 FOR I=0 TO 2 STEP 1/8
  15 IF I=1/2 OR I=3/2 THEN GOTO
 30
  20 PRINT "TAN(PI*";I;")","= ";
TAN (PI*I)
  30 NEXT I
```

Figure T.3 : TAN-Exemple de programme

```
      LA FONCTION TANGENTE
      ====================

TAN(PI*0)           = 0
TAN(PI*0,125)       = 0.41421356
TAN(PI*0,25)        = 1
TAN(PI*0,375)       = 2,4142136
TAN(PI*0,625)       = -2,4142136
TAN(PI*0,75)        = -1
TAN(PI*0,875)       = -0,41421356
TAN(PI*1)           = 0
TAN(PI*1,125)       = 0.41421356
TAN(PI*1,25)        = 1
TAN(PI*1,375)       = 2,4142136
TAN(PI*1,625)       = -2,4142136
TAN(PI*1,75)        = -1
TAN(PI*1,875)       = -0,41421356
TAN(PI*2)           = 0
```

Figure T.4 : TAN-Exemple de résultat

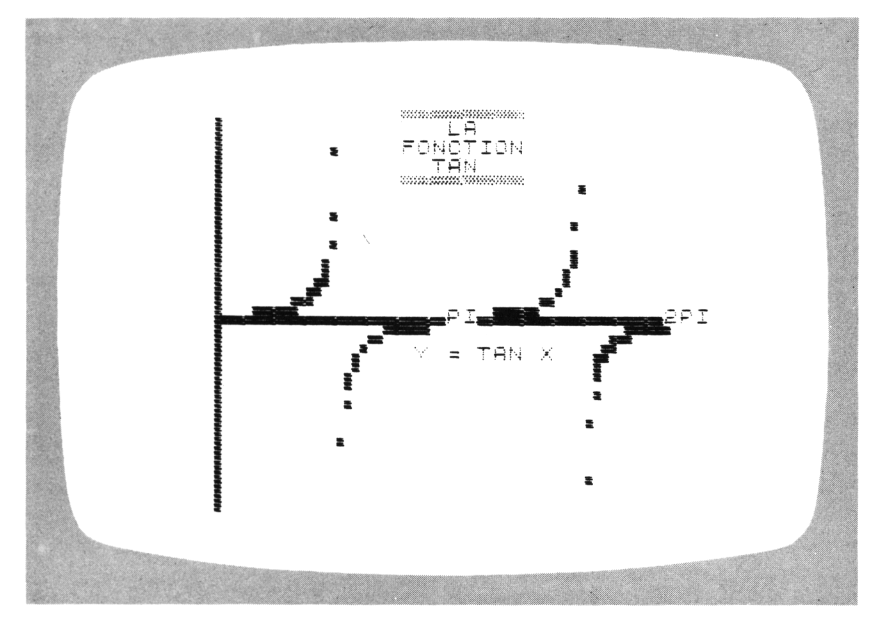


Figure T.5 : TAN-Tracé de courbe

trôle I de la boucle FOR (ligne 10) détermine les arguments de la fonction TAN, ligne 20. La ligne 15 saute les arguments $\pi/2$ et $3\pi/2$. On voit le résultat de ce programme Figure T.4.

Notes et commentaires

- La Figure T.5 montre la courbe de la fonction tangente pour x variant entre -2π et 2π. Cette courbe est générée par la commande PLOT du ZX 81.

- Pour plus d'information sur les fonctions trigonométriques, voir les rubriques SIN et COS.

THEN (shift ; touche [3])

Chaque instruction IF doit être accompagnée d'une instruction THEN. Si l'expression logique suivant le mot IF est vraie, l'ordina-

teur exécute le mot clé qui suit toujours l'instruction THEN. Pour plus d'information, voir la rubrique IF.

TO (shift ; touche [4])

Le mot clé TO a deux utilisations distinctes en BASIC ZX 81 :

1. L'instruction TO est associée à la syntaxe d'une boucle FOR. Elle indique les limites de la variable de la boucle de contrôle, par exemple :

   FOR I=1 **TO** 30

   Pendant l'exécution de la boucle FOR, la variable I est incrémentée de 1 à 30. Pour plus d'information, voir les rubriques FOR et STEP.

2. L'instruction TO permet l'identification, l'accès ou l'assignation d'une *portion* de chaîne ; cette technique est appelée *découpage* de chaîne. On prend l'expression suivante

   S$(M **TO** N)

   dans laquelle M et N sont des valeurs numériques (valeurs littérales, variables ou expressions numériques) comprises entre 1 et LEN S$. Si N est supérieure ou égale à M, cette expression identifie une *sous-chaîne* de S$ composée de caractères compris entre le M-ième et le N-ième inclus. Si M est supérieure à N, la sous-chaîne est vide. Si N est supérieure à LEN S$, l'expression n'est pas légale ; l'ordinateur affiche un message d'erreur "3".

Pour étudier quelques exemples, on imagine que S$ contient une chaîne de onze caractères :

LET S$="INTERESSANT"

Les instructions :

```
10 PRINT S$(5 TO 8)
20 PRINT S$(1 TO 8)
30 PRINT S$(5 TO 11)
```

affichent respectivement les sous-chaînes RESS, INTERESS, RESSANT. On peut abréger les deux dernières instructions de découpage comme suit :

```
20 PRINT S$(TO 8)
30 PRINT S$(5 TO)
```

Etant donné que la sous-chaîne de la ligne 20 commence au premier caractère de S$, le nombre qui *précède* TO peut être omis. De même, puisque la sous-chaîne de la ligne 30 se termine au dernier caractère de S$, le nombre qui suit TO peut être supprimé.

L'instruction :

```
40 PRINT S$(6 TO 12)
```

entraîne un arrêt du programme avec un message d'erreur parce que S$ ne contient que onze caractères.

Enfin, on peut également utiliser la technique du découpage pour *modifier* une portion de chaîne. Par exemple, si la longueur de S$ n'est pas modifiée, l'instruction :

```
50 LET S$(1 TO 6) = "ENVAHI"
```

change la valeur de S$ en ENVAHISSANT. Cette instruction assigne les caractères ENVAHI aux six premiers éléments de S$.

Exemple de programme

Le programme des Figures T.6 et T.7 illustre la technique de découpage de chaîne. Ce programme est une extension du programme de mélange de cartes décrit dans la rubrique RND (Figure R.3). Dans ce programme plus simple, les cartes battues étaient représentées sur l'écran par des nombres compris entre 1 et 52. Cette version remplace les nombres par les noms des cartes

```
 10 PRINT " MELANGE D UN JEU DE
52 CARTES"
 15 PRINT
 20 DIM D(52)
 30 FOR I=1 TO 52
 40 LET D(I)=I
 50 NEXT I
 60 FOR I=1 TO 52
 70 LET R=INT (52*RND)+1
 80 LET H=D(I)
 90 LET D(I)=D(R)
100 LET D(R)=H
110 NEXT I
120 LET C$="COEUR  CARREAUTREFL
E PIQUE  "
130 LET V$="AS    DEUX  TROIS Q
UATRECINQ  SIX   SEPT  HUIT  NEU
F  DIX   VALET DAME  ROI   "
140 FOR I=1 TO 52
150 PRINT "   ";D(I);TAB 6;": "
160 GOSUB 200
```

Figure T.6 : TO-Exemple de programme

```
170 NEXT I
180 STOP
200 REM ** DECOUPAGE DE LA
210 REM ** COULEUR DANS C$ ET
220 REM ** DE LA VALEUR DANS V$
230 REM
240 LET CC=(INT ((D(I)-1)/13)+1
)*7
250 LET VC=(D(I)-INT ((D(I)-1)/
13)*13)*6
260 PRINT "  ";V$(VC-5 TO VC);"
DE ";C$(CC-6 TO CC)
270 RETURN
```

Figure T.7 : TO-Exemple de programme, (suite)

(valeur et couleur). La Figure T.8 donne les premiers résultats de ce programme.

Cette version est identique à la première jusqu'à la ligne 110. Le tableau D représente le paquet de cartes. La boucle FOR, lignes 60 à 110, mélange le jeu.

Les nouvelles instructions qui affichent le jeu sur l'écran commencent à la ligne 120. Deux longues chaînes sont assignées aux variables S$ et R$; S$ contient les noms des couleurs et R$ les noms des valeurs. Dans S$, chaque nom est exactement composé de sept caractères ; dans R$, chaque nom est exactement composé de cinq caractères (les noms plus courts sont complétés par des espaces). Ces conditions sont indispensables à la technique de découpage.

La boucle FOR de la ligne 140 affiche le numéro de chaque carte battue, puis appelle le sous-programme de la ligne 200 pour qu'il affiche sa valeur et sa couleur.

Le rôle du sous-programme (Figure T.7) est de déterminer des sous-chaînes de S$ et de R$ pour chaque nombre compris entre 1 et 52. Les couleurs sont organisées comme suit :

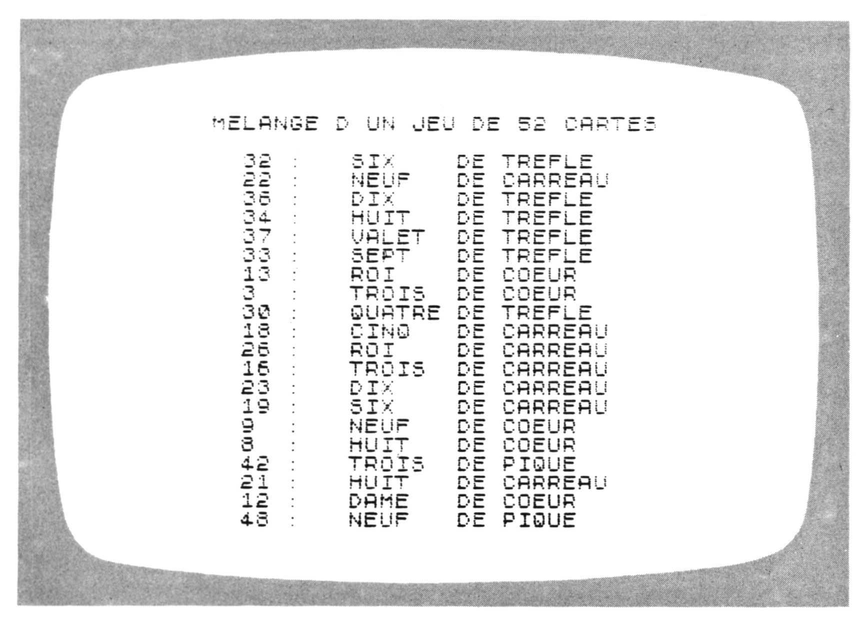


Figure T.8 : TO-Exemple de résultat

cartes 1 à 13 : cœurs
cartes 14 à 26 : carreaux
cartes 27 à 39 : trèfles
cartes 40 à 52 : piques

A l'intérieur de chaque couleur, l'as est la première carte et les figures sont les trois dernières cartes. La traduction des nombres en couleurs et valeurs nécessite des opérations arithmétiques qui ne peuvent être exécutées automatiquement par les fonctions intégrées au BASIC du ZX 81. Il faut, pour les couleurs, trouver pour chaque carte un nombre compris entre 1 et 4 ; ce nombre est le résultat de ce que certains langages de programmation appellent DIV (division entière). Il faut, pour les valeurs, trouver pour chaque carte un nombre compris entre 1 et 13 ; le modulo (appelé MOD dans certains langages), *reste* d'une division, est l'opération qui fournit ce nombre. Ces deux opérations sont respectivement développées aux lignes 240 et 250 ; elles assignent des valeurs entières à CC (couleur de la carte) et VC (valeur de la carte).

La variable CC est un index de la chaîne C$; elle contient un multiple de 7, compris entre 7 et 28. L'expression :

```
C$(CC-6 TO CC)
```

fournit donc le nom d'une des couleurs.

La variable VC est un index de la chaîne V$; elle contient un multiple de 5, compris entre 5 et 65. L'expression :

```
V$(VC-5 TO VC)
```

fournit donc le nom d'une des valeurs.

La ligne 260 utilise ces deux expressions de découpage pour afficher l'intitulé complet d'une carte.

UNPLOT (mot clé ; touche [W])

La commande UNPLOT efface un point graphique situé sur un emplacement déterminé de l'écran. La forme générale de l'instruction UNPLOT est :

```
  1 FAST
  2 FOR X=0 TO 63
  3 FOR Y=0 TO 43
  5 PLOT X,Y
  7 NEXT Y
  9 NEXT X
 10 LET V=21
 20 LET H=0
 25 SLOW
 30 PRINT AT V,H;"R  "
 40 INPUT R
 50 PRINT AT V,H;"C H"
 60 INPUT CH
 70 PRINT AT V,H;"C V"
 80 INPUT CV
 85 FAST
 90 FOR A=0 TO 2*PI STEP PI/R
100 LET X=R*COS A+CH
110 LET Y=R*SIN A+CV
120 UNPLOT X,Y
130 NEXT A
140 GOTO 25
```

Figure U.1 : UNPLOT-Exemple de programme

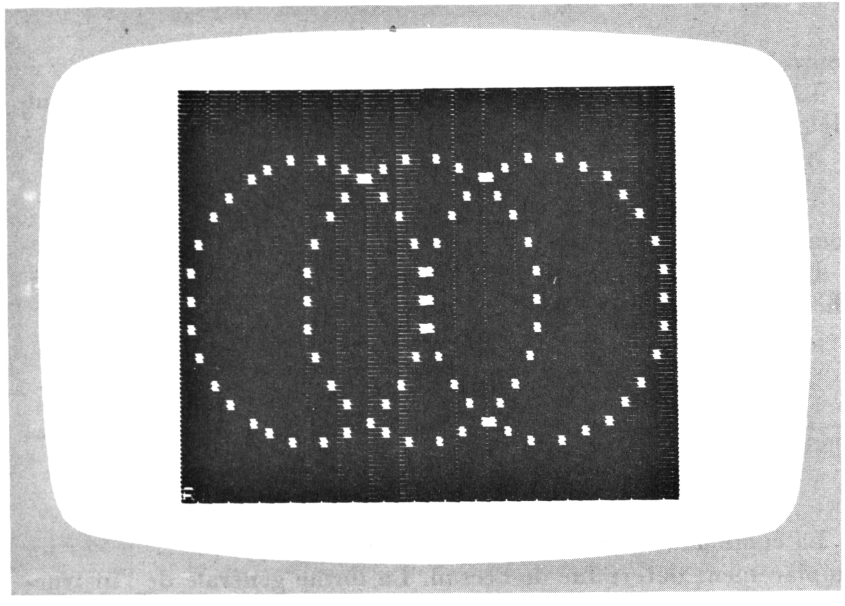


Figure U.2 : UNPLOT-Exemple de résultat

UNPLOT H,V

dans laquelle H et V sont respectivement les coordonnées d'adresses horizontales et verticales de l'instruction UNPLOT. H et V peuvent être exprimées sous la forme de valeurs numériques littérales, de variables ou d'expressions arithmétiques.

L'utilisation de l'instruction UNPLOT est résumée dans la Figure P.4 de la rubrique PLOT.

Exemple de programme

La Figure U.1 montre un programme qui illustre l'utilisation de l'instruction UNPLOT. Ce programme, comme celui qui est décrit dans la rubrique PLOT, dessine des cercles sur l'écran. Mais dans le cas présent, les cercles sont constitués de points blancs sur fond sombre, comme le montre la Figure U.2.

Les lignes 2 à 9 de ce programme utilisent, pour noircir l'écran, une commande PLOT insérée dans deux boucles imbriquées. Puis les lignes 30 à 80 reçoivent du clavier trois informations qui définissent le cercle désiré: "R:" pour le rayon (en points graphiques); "C:H" pour la coordonnée horizontale du centre et "C:V" pour sa coordonnée verticale. Ensuite, les lignes 90 à 130 dessinent le cercle en se servant d'une formule qui utilise les fonctions COS et SIN. La commande UNPLOT de la ligne 120 génère le cercle blanc sur fond noir.

Notes et commentaires

- Pour plus d'information, voir la rubrique PLOT.

USR (fonction ; touche [L])

La fonction USR appelle un programme en code machine qui se trouve à une adresse précise dans la mémoire de l'ordinateur. Pour le ZX 81, un programme en code machine est un ensemble de commandes écrit avec le jeu d'instruction du microprocesseur Z80. La

fonction USR permet l'exécution de ce type de programme pendant le déroulement d'un programme BASIC. Cette fonction prend la forme :

USR A

dans laquelle A est l'adresse mémoire du début du programme. La fonction USR donne la valeur des deux registres bc.

Pour plus d'information, voir la rubrique POKE.

VAL (fonction ; touche [J])

La fonction VAL donne l'équivalent numérique d'une chaîne. Elle prend la forme :

VAL S$

dans laquelle S$ est une chaîne qui peut être convertie en un nombre. La valeur de S$ peut prendre la forme d'une des combinaisons suivantes :

1. Chiffres, sous forme de caractères
2. Fonctions numériques légales
3. Expressions arithmétiques
4. Noms de variables numériques définies auparavant.

Par exemple, toutes les chaînes suivantes sont des arguments acceptés pour la fonction VAL :

```
"9862.89"
"(18+15)*22"
"INT   (17/2.5)"
"X*COS   Y"
```

On remarque que les noms de fonction (INT et COS dans ces exemples) ne peuvent pas être tapés lettre par lettre ; pour qu'elles soient "légales", on doit les entrer à l'aide des touches fonctions. D'autre part, les variables X et Y du dernier exemple doivent avoir été définies avant d'être utilisées comme arguments de la fonction VAL.

Si on attribue à cette fonction un argument qui ne peut pas être converti en une valeur numérique, l'ordinateur arrête l'exécution du programme et donne un message d'erreur "C".

Exemple de programme

La Figure V.1 montre un programme de validation d'entrée numérique. Il peut être utilisé pour prévenir les erreurs de frappe dans des programmes qui nécessitent de nombreuses entrées numériques. Quand l'ordinateur exécute une instruction INPUT pour assigner une valeur à une variable numérique, il réagit de deux façons différentes si la valeur entrée contient des éléments non numériques. Si le premier caractère entré est une lettre et que les autres sont des

```
  10 LET T=1
  20 LET F=NOT T
  30 PRINT AT 21,0;"NOMBRE :"
  40 GOSUB 100
  50 LET N=VAL N$
  60 CLS
  70 PRINT AT 9,9;"NOMBRE = ";N
  80 GOTO 30
  99 REM ** VALIDATION D ENTREE
 100 LET BON=T
 105 INPUT N$
 107 IF N$="" THEN GOTO 105
 110 FOR I=1 TO LEN N$
 120 IF N$(I)<"0" OR N$(I)>"9"
THEN LET BON=F
 130 NEXT I
 140 IF BON THEN RETURN
 150 PRINT AT 21,0;"**RECOMMENCE
R"
 160 GOTO 100
```

Figure V.1 : VAL-Exemple de programme

chiffres, le programme s'arrête avec un message d'erreur "2". Cela signifie que l'ordinateur a identifié cette entrée comme un nom de variable numérique au lieu de l'identifier comme une valeur numérique littérale. Il indique que cette variable n'a pas encore été définie (voir rubrique INPUT). Si les données entrées contiennent des caractères illégaux autres que le premier caractère, l'ordinateur refuse l'entrée. Le repère S (erreur de syntaxe) apparaît dans l'espace de travail pour montrer où se trouve l'erreur.

Quand on doit entrer beaucoup de nombres dans un programme, on peut utiliser la méthode de programmation suivante pour éviter les ennuis précédents :

1. Ecrire une instruction INPUT qui lit les valeurs entrées dans une variable de chaîne plutôt que dans une variable numérique.

2. Envoyer la chaîne à un programme prévu pour vérifier que chaque caractère qui la compose est un chiffre compris entre 0 et 9.

3. Une fois la chaîne validée, utiliser la fonction VAL pour la convertir en une valeur numérique.

Le programme de la Figure V.1 utilise cette méthode. La ligne 30 affiche sur l'écran un signal réclamant l'entrée de données. La ligne 40 appelle le sous-programme de validation d'entrée (ligne 100) qui lit à plusieurs reprises les valeurs entrées dans la chaîne N$, et renvoie le contrôle au programme principal lorsque la chaîne N$ ne contient plus que des chiffres. La ligne 50 assigne alors l'équivalent numérique de N$ à la variable N :

```
50 LET N=VAL N$
```

Le sous-programme d'entrée (ligne 100) établit une "variable logique" appelée BON pour indiquer si la valeur entrée est légale ou non (pour avoir des explications supplémentaires sur ce type de variables, voir la rubrique IF). Ce sous-programme lit la chaîne N$ (ligne 105) et la teste pour vérifier qu'elle n'est pas vide (ligne 107). Il examine ensuite, à l'intérieur d'une boucle FOR, chaque caractère de N$, de 1 à LEN N$:

```
100 FOR I=1 TO LEN N$
```

Si un caractère quelconque N$(I) est en dehors des limites légales, c'est-à-dire entre "0" et "9", le sous-programme attribue à la variable BON la valeur F (faux):

120 IF N$(I) < "0" **OR** N$(I) > "9" **THEN LET** BON = F

Les variables T et F correspondant à "vrai" et "faux" sont initialisées en début de programme. Si BON est encore vrai au moment où la boucle a terminé l'examen de la chaîne N$, le sous-programme peut retourner au programme principal:

140 **IF** BON **THEN RETURN**

Sinon, il affiche le message d'erreur suivant:

**RECOMMENCER

et retourne à la ligne 100 pour l'introduction d'une nouvelle valeur.

On peut lancer le programme et essayer d'entrer des valeurs illégales pour voir ce que cela entraîne. La version de ce programme ne lit que des valeurs entières. Pour introduire des valeurs réelles, on peut modifier la ligne 120 comme suit:

120 **IF** (N(I) < "0" **OR** N$(I) > "9")
AND N(I) < > "." **THEN LET** BON = F

Dans cette version, on considère le point décimal comme un caractère légal. Pour parfaire le programme, on peut ajouter un test qui exclue les valeurs contenant plus d'un point décimal.

Valeur littérale (vocabulaire informatique)

Une valeur littérale est une valeur numérique ou une valeur de chaîne entrée comme une constante dans une instruction de

programme, ou comme une réponse à une demande d'entrée de données (INPUT). Inversement, un nom de variable *représente* les valeurs numériques ou de chaîne qui sont stockées sous ce nom dans la mémoire de l'ordinateur. Une valeur de chaîne littérale apparaît entre guillemets dans une instruction. Par exemple, l'instruction suivante assigne une valeur de chaîne littérale à la variable S$:

```
LET S$="ORDINATEUR"
```

Une valeur numérique littérale peut se présenter sous forme de notation décimale ou de notation scientifique, comme dans les exemples suivants :

```
LET N1=123,455
LET N2=3,45E10
```

Variable (vocabulaire informatique)

Une variable est un emplacement réservé à un certain type de donnée dans la mémoire de l'ordinateur, et représenté par un certain nom dans un programme. Les variables du BASIC du ZX 81 sont de deux sortes, elles représentent deux types de données. Le nom de la variable indique son genre :

- Le nom d'une variable de chaîne est constitué d'une lettre suivie du caractère $, par exemple : S$, L$, T$.
- Les noms de variables numériques peuvent avoir n'importe quelle longueur, mais le premier caractère doit être une lettre. Les autres caractères peuvent être des lettres ou des chiffres, par exemple :

```
N
V1
V2
MAINTENIR
VALEUR5
```

Il y a deux exceptions à cette règle : le nom de la variable de contrôle d'une boucle FOR ne doit comporter qu'une seule lettre, de même pour le nom d'un tableau numérique (voir les rubriques DIM, LET et FOR).

LA BIBLIOTHÈQUE SYBEX

OUVRAGES GÉNÉRAUX

VOTRE PREMIER ORDINATEUR *par RODNAY ZAKS,*
296 pages, Réf. 226

VOTRE ORDINATEUR ET VOUS *par RODNAY ZAKS,*
296 pages, Réf. 271

DU COMPOSANT AU SYSTÈME : une introduction aux microprocesseurs *par RODNAY ZAKS,*
636 pages, Réf. 239

TECHNIQUES D'INTERFACE aux microprocesseurs *par AUSTIN LESEA ET RODNAY ZAKS,*
450 pages, Réf. 230, 3ème édition

LEXIQUE INTERNATIONAL MICROORDINATEURS, avec dictionnaire abrégé en 10 langues
192 pages, Réf. 234

BASIC

VOTRE PREMIER PROGRAMME BASIC *par RODNAY ZAKS,*
208 pages, Réf. 263

INTRODUCTION AU BASIC *par PIERRE LE BEUX,*
336 pages, Réf. 216

LE BASIC PAR LA PRATIQUE : 60 exercices *par JEAN-PIERRE LAMOITIER,*
252 pages, Réf. 231

LE BASIC POUR L'ENTREPRISE *par XUAN TUNG BUI,*
204 pages, Réf. 253, 2ème édition

PROGRAMMES EN BASIC, Mathématiques, Statistiques, Informatique *par ALAN R. MILLER,*
318 pages, Réf. 259

AU COEUR DES JEUX EN BASIC *par RICHARD MATEOSIAN,*
352 pages, Réf. 233

JEUX D'ORDINATEUR EN BASIC *par DAVID H. AHL,*
192 pages, Réf. 24€

NOUVEAUX JEUX D'ORDINATEUR EN BASIC *par DAVID H. AHL,*
204 pages, Réf. 247

PASCAL

INTRODUCTION AU PASCAL *par PIERRE LE BEUX,*
496 pages, Réf. 222

LE PASCAL PAR LA PRATIQUE *par Pierre Le Beux et Henri Tavernier,*
562 pages, Réf. 229

LE GUIDE DU PASCAL *par Jacques Tiberghien,*
504 pages, Réf. 232

PROGRAMMES EN PASCAL pour Scientifiques et Ingénieurs *par Alan R. Miller,*
392 pages, Réf. 240

AUTRES LANGAGES

INTRODUCTION A ADA *par Pierre Le Beux,*
366 pages, Réf. 242

MICROORDINATEUR

ALICE

JEUX EN BASIC POUR ALICE *par Pierre Monsaut,*
96 pages, Réf. 320

APPLE

PROGRAMMEZ EN BASIC SUR APPLE II *par Léopold Laurent,*
208 pages, tome 1, Réf. 268

APPLE II 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 283

JEUX EN PASCAL SUR APPLE *par Douglas Hergert et Joseph T. Kalash,*
372 pages, Réf. 241

ATARI

JEUX EN BASIC SUR ATARI *par Paul bunn,*
96 pages, Réf. 282

COMMODORE

JEUX EN BASIC SUR COMMODORE 64 *par Pierre Monsaut,*
96 pages, Réf. 317

GOUPIL

PROGRAMMEZ VOS JEUX SUR GOUPIL *par François Abella,*
208 pages, Réf. 264

IBM

EXERCICES EN BASIC SUR L'ORDINATEUR PERSONNEL IBM *par Jean-Pierre Lamoitier,*
256 pages, Réf. 267

IBM PC Guide de l'utilisateur, *par Joan Lasselle et Carol Ramsey,*
160 pages, Réf. 301

IBM PC 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 280

ORIC

JEUX EN BASIC SUR ORIC *par Peter Shaw,*
96 pages, Réf. 278

SHARP

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2 *par Michel Lhoir,*
2 tomes, Réf. 261-262

SPECTRUM

PROGRAMMEZ EN BASIC SUR SPECTRUM *par S.M. Gee,*
208 pages, Réf. 252

JEUX EN BASIC SUR SPECTRUM *par Peter Shaw,*
96 pages, Réf. 276

TIMEX

DÉCOUVREZ LE ZX 81 ET LE TIMEX SINCLAIR 1000 *par Douglas Hergert,*
208 pages, Réf. 256

TRS-80

PROGRAMMEZ EN BASIC SUR TRS-80 *par Léopold Laurent,*
2 tomes, Réf. 250-251

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2 *par Michel Lhoir,*
2 tomes, Réf. 261-262

JEUX EN BASIC SUR TRS-80 MC-10 *par Pierre Monsaut,*
96 pages, Réf. 323

JEUX EN BASIC SUR TRS-80 *par Chris Palmer,*
96 pages, Réf. 302

VIC 20

PROGRAMMEZ EN BASIC SUR VIC 20 *par G. O. Hamann,*
2 tomes, Réf. 244-245

JEUX EN BASIC SUR VIC 20 *par Alastair Gourlay,*
96 pages, Réf. 277

ZX 81

DÉCOUVREZ LE ZX 81 ET LE TIMEX SINCLAIR 1000 *par Douglas Hergert,*
208 pages, Réf. 256

ZX 81 56 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 304

JEUX EN BASIC SUR ZX 81 *par Mark Charlton,*
96 pages, Réf. 275

MICRO-PROCESSEURS

PROGRAMMATION DU Z80 *par Rodnay Zaks,*
618 pages, Réf. 220

APPLICATIONS DU Z80 *par James W. Coffron,*
304 pages, Réf. 274

PROGRAMMATION DU 6502 *par Rodnay Zaks,*
376 pages, Réf. 224, 2ème édition

APPLICATIONS DU 6502 *par Rodnay Zaks,*
288 pages, Réf. 219

PROGRAMMATION DU 6800 *par Daniel-Jean David et Rodnay Zaks,*
374 pages, Réf. 218

PROGRAMMATION DU 6809 *par Rodnay Zaks et William Labiak,*
392 pages, Réf. 270

SYSTÈMES D'EXPLOITATION

GUIDE DU CP/M AVEC MP/M *par Rodnay Zaks,*
354 pages, Réf. 228

CP/M APPROFONDI *par Alan R. Miller,*
380 pages, Réf. 265

INTRODUCTION AU p-SYSTEM UCSD *par Charles W. Grant et Jon Butah,*
308 pages, Réf. 257

LOGICIELS ET APPLICATIONS

INTRODUCTION AU TRAITEMENT DE TEXTE *par Hal Glatzer,*
228 pages, Réf. 243

INTRODUCTION A WORDSTAR *par Arthur Naiman,*
200 pages, Réf. 255

VISICALC APPLICATIONS *par Stanley R. Trost,*
304 pages, Réf. 258

La plupart de ces ouvrages existent en version anglaise. N'hésitez pas à demander notre catalogue.

EN ANGLAIS

BASIC EXERCISES FOR APPLE *by* JEAN-PIERRE LAMOITIER,
232 pages, Réf. 0-084

BASIC FOR BUSINESS *by* DOUGLAS HERGERT,
224 pages, Réf. 0-080

CELESTIAL BASIC : Astronomy on your Computer *by* ERIC BURGESS,
228 pages, Réf. 0-087

INTRODUCTION TO PASCAL (Including UCSD Pascal) *by* RODNAY ZAKS,
422 pages, Réf. 0-066

DOING BUSINESS WITH PASCAL *by* RICHARD HERGERT AND DOUGLAS HERGERT,
380 pages, Réf. 0-091

MASTERING VISICALC *by* DOUGLAS HERGERT,
224 pages, Réf. 0-090

THE APPLE CONNECTION *by* JAMES W. COFFRON,
228 pages, Réf. 0-085

PROGRAMMING THE Z8000 *by* RICHARD MATEOSIAN,
300 pages, Réf. 0-032

A MICROPROGRAMMED APL IMPLEMENTATION *by* RODNAY ZAKS,
350 pages, Réf. 0-005

ADVANCED 6502 PROGRAMMING *by* RODNAY ZAKS,
292 pages, Réf. 0-089

FORTRAN PROGRAMS FOR SCIENTISTS AND ENGINEERS *by* ALAN R. MILLER,
320 pages, Réf. 0-082

POUR UN CATALOGUE COMPLET DE NOS PUBLICATIONS

FRANCE
6-8, impasse du Curé
75018 Paris
Tél. : (1) 203.95.95
Telex : 211801 F

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tél. : (415) 848.8233
Télex : 336311

ALLEMAGNE
Volgelsanger. Weg 111
4000 Düsseldorf 30
Post Bos N° 30-09-61
Tél. : (0211) 626441
Telex : 08588163



Imprimerie Sagi – 9-11, rue du Roule, 75001 Paris 206/04540

GUIDE DU BASIC

ZX 81

Ce guide est un dictionnaire exhaustif du BASIC du ZX 81 permettant au débutant comme au programmeur expérimenté de tirer le meilleur parti de son micro-ordinateur. Chaque instruction, commande et fonction est présentée, commentée et illustrée par des exemples de programmes. L'étude de chacun de ces exemples permettra de comprendre et de mieux exploiter les nombreuses possibilités du ZX 81. Certains de ces programmes pourront même être utilisés directement ou intégrés à des programmes plus importants. Les principaux termes du vocabulaire informatique sont également définis et illustrés par des exemples d'applications.

L'AUTEUR

Douglas HERGERT est l'auteur de nombreux livres publiés par SYBEX : en France (Découvrez le ZX 81, Guide du BASIC Apple II) et aux Etats-Unis (BASIC for Business, Mastering VisiCalc). Il est également coauteur de Jeux en Pascal sur Apple et Doing Business with Pascal.

ISBN 2-902414-85-4   285   02-84   78 F


SYBEX
Douglas Hergert GUIDE DU BASIC ZX 81



Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE



https://acpc.me/